
Prefazione

Benvenuti nel mondo del C++! Questo libro è stato scritto da un padre, Harvey M. Deitel, e da suo figlio, Paul J. Deitel. Il padre ha programmato e/o insegnato la programmazione per 38 anni, mentre il figlio ha programmato e/o insegnato la programmazione per 18. Il padre programma e insegna grazie all'esperienza maturata; il figlio, invece, programma e insegna grazie a un'inesauribile riserva di energia. Il padre punta alla chiarezza, il figlio alle prestazioni. Il padre cerca l'eleganza e la bellezza, il figlio vuole soprattutto risultati concreti. Insieme hanno cercato di realizzare un libro che speriamo possiate trovare utile, completo e divertente.

Lo scopo di questo libro

Il Professor Harvey M. Deitel ha tenuto corsi universitari introduttivi sulla programmazione per 20 anni, puntando soprattutto a insegnare come sia possibile sviluppare programmi ben scritti e ben strutturati. Molti dei suoi insegnamenti riguardano i concetti fondamentali della programmazione, con particolare enfasi sull'utilizzo efficace delle strutture di controllo e sulla funzionalità. Questi argomenti vengono presentati all'interno di questo libro esattamente nel modo in cui il Professor Deitel li ha sempre proposti ai propri studenti universitari.

In base alla nostra esperienza, gli studenti affrontano i temi presentati in questi capitoli nello stesso modo in cui affrontano i corsi di Pascal. C'è, comunque, un'importante differenza: gli studenti sono estremamente motivati dal fatto che stanno imparando un linguaggio che potranno utilizzare immediatamente quando lasceranno l'università e affronteranno il mondo del lavoro. Tutto questo aumenta notevolmente il loro entusiasmo nei confronti del C++, nonostante ci sia moltissimo da imparare.

Il nostro obiettivo era chiaro: offrire un testo sul C++ destinato ai corsi universitari a livello introduttivo, per quegli studenti che non hanno alcuna esperienza nell'ambito della programmazione, pur fornendo, allo stesso tempo, la completezza teorica e pratica richiesta ai tradizionali corsi avanzati di C++. Con questo proposito abbiamo scritto i due volumi *C++ Fondamenti di programmazione* e *C++ Tecniche avanzate di programmazione*; nell'insieme, un'opera assai completa su questo linguaggio.

Un importante elemento caratterizzante il testo è UML (Unified Modeling Language), uno strumento di rappresentazione grafica dei sistemi che utilizziamo nell'analisi completa, dalla definizione all'implementazione, di un caso complesso di progettazione orientata agli oggetti. La nostra sensazione è che i testi introduttivi di programmazione trascurino i progetti orientati ad oggetti di portata più vasta, quindi vogliamo raccomandare lo studio di questo caso opzionale, perché migliorerà sensibilmente l'approccio alla progettazione degli studenti del primo anno. Essi, infatti, avranno l'opportunità di navigare da subito in più di 1000 linee di codice C++ scritte e analizzate attentamente da revisori accademici e professionisti del settore.

Alla fine dei primi sette capitoli e del Capitolo 9 troverete una sezione intitolata “Pensare in termini di oggetti”, il cui scopo è presentare gradualmente la progettazione orientata agli oggetti per mezzo di UML. UML è oggi lo schema più utilizzato per rappresentare graficamente i sistemi orientati agli oggetti. UML è un linguaggio grafico complesso e prevede numerose funzionalità, di cui noi vogliamo presentare soltanto un sottoinsieme. Le funzionalità che analizziamo servono per guidare il lettore nella sua prima esperienza di progettazione, rivolta per l'appunto a programmatori principianti. Lo studio di questo progetto ci accompagna per diversi capitoli fino alla sua implementazione, per cui esitiamo a chiamarlo soltanto un esercizio, perché è piuttosto un'esperienza di programmazione che si conclude solo con l'analisi del codice C++ completo.

Nei primi cinque capitoli del testo ci concentriamo sulla metodologia “convenzionale” che prende il nome di programmazione strutturata, perché gli oggetti che creeremo in seguito saranno composti in parte da porzioni di codice strutturato. Ognuno di questi capitoli termina con una sezione intitolata “Pensare in termini di oggetti”, in cui presentiamo un'introduzione alla programmazione orientata agli oggetti utilizzando UML (Unified Modeling Language). Queste sezioni hanno lo scopo di aiutare gli studenti a sviluppare una mentalità orientata agli oggetti, in modo tale che sappiano impiegare da subito i concetti di programmazione orientata agli oggetti illustrati a partire dal Capitolo 6. Nella sezione “Pensare in termini di oggetti” del Capitolo 1 introduciamo i concetti fondamentali e la terminologia, mentre nelle sezioni analoghe dei Capitoli dal 2 al 5 affrontiamo questioni più sostanziali, affrontando un problema complesso con le tecniche di progettazione orientata agli oggetti (OOD, dall'inglese Object Oriented Design). In queste sezioni analizziamo la tipica definizione di un problema, che specifica il sistema desiderato, e determiniamo gli oggetti da implementare, gli attributi ad essi necessari, i comportamenti che dovranno esibire e il tipo di interazione che dovranno avere tra loro perché il sistema funzioni come specificato. Questi discorsi vengono prima di qualsiasi cenno alla programmazione orientata agli oggetti in C++. Nelle sezioni “Pensare in termini di oggetti” dei Capitoli 6, 7 e 9 discutiamo l'implementazione in C++ del sistema orientato agli oggetti che avremo progettato nei capitoli precedenti.

Questo è il progetto più complesso ed esteso del libro, e noi pensiamo che per gli studenti possa costituire un'esperienza significativa di progettazione e implementazione. L'estensione del progetto ci ha costretto a includere argomenti che non riprendiamo in nessun'altra sezione del libro, tra cui l'interazione tra oggetti, gli handle, pregi e difetti di riferimenti e di puntatori e infine le dichiarazioni anticipate, che consentono di evitare il problema dell'inclusione circolare dei file. Questo progetto rivelerà la sua utilità agli studenti quando dovranno affrontare i reali problemi dell'industria.

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.

Le sezioni “Pensare in termini di oggetti”

Nel Capitolo 2 iniziamo la prima fase di progettazione orientata agli oggetti (OOD) del simulatore di ascensore identificando le classi che lo implementeranno. Introduciamo inoltre UML, con i casi d’uso, i diagrammi delle classi e degli oggetti e i concetti di associazione, molteplicità, composizione, ruolo e collegamento.

Nel Capitolo 3 determiniamo la maggior parte degli attributi necessari alle classi che abbiamo individuato. Continuiamo inoltre a introdurre UML, con i diagrammi di stato e delle attività, insieme con i concetti di evento e azione e la loro correlazione ai suddetti diagrammi.

Nel Capitolo 4 determiniamo la maggior parte delle operazioni (comportamenti) delle classi individuate. Introduciamo inoltre i diagrammi di sequenza di UML e il concetto di “messaggio inviato a un oggetto”.

Nel Capitolo 5 determiniamo la maggior parte delle collaborazioni (interazioni tra oggetti) che servono a implementare il sistema, rappresentandole anche con i diagrammi delle collaborazioni UML. A fine sezione abbiamo inserito una bibliografia e un elenco di risorse in rete su UML, che mettono a disposizione tra le altre cose le specifiche di UML 1.3 e materiali come tutorial, FAQ, articoli e software.

Nel Capitolo 6 utilizziamo il diagramma delle classi UML sviluppato nelle sezioni precedenti per determinare i file di intestazione C++ che definiranno le classi. Introduciamo inoltre il concetto di *handle a un oggetto* e ne iniziamo a studiare l’implementazione in C++.

Nel Capitolo 7 presentiamo il codice C++ completo del programma di simulazione, circa 1000 linee di codice, e lo analizziamo in dettaglio. Il codice deriva direttamente dalla progettazione UML delle sezioni precedenti e tiene conto di tutte le “buone abitudini di programmazione” che predichiamo, tra cui l’utilizzo di dati membro e funzioni **static** e **const**. Parliamo inoltre dell’allocazione dinamica della memoria, della composizione e dell’interazione tra oggetti per mezzo degli handle, ed infine del problema dell’inclusione circolare dei file che si può evitare grazie alle dichiarazioni anticipate.

Nel Capitolo 9 introduciamo il concetto di ereditarietà nel programma presentato nel Capitolo 7. Sugeriamo anche ulteriori miglioramenti al codice, ma li lasciamo agli studenti come esercizio, perché possano cimentarsi con gli strumenti di progettazione con cui avranno familiarizzato nelle sezioni precedenti.

Speriamo sinceramente che questo progetto possa essere un’esperienza di studio stimolante e utile per studenti e docenti. Da parte nostra abbiamo cercato di rendere l’intero procedimento di progettazione il più graduale possibile. Inoltre il programma che presentiamo utilizza diverse nozioni chiave della programmazione, tra cui le classi, gli oggetti, l’incapsulamento, la visibilità, la composizione e l’ereditarietà. Vi saremo grati per tutti i commenti, le critiche e i suggerimenti che vorrete inviarci all’indirizzo **deitel@deitel.com**.

II CD-ROM

- Tutti i *programmi di esempio* sono contenuti nel CD-ROM allegato al volume. Ciò consente ai docenti di preparare le lezioni e agli studenti di padroneggiare il linguaggio più rapidamente. Gli esempi si possono anche scaricare dal sito www.deitel.com o dal booksite abbinato a questo libro (www.apogeeonline.com/education/booksite). Per estrarre i sorgenti dai file .ZIP occorre utilizzare un programma di decompressione come WinZip ([http:// www.winzip.com/](http://www.winzip.com/)) o PKZIP ([http:// www.pkware.com/](http://www.pkware.com/)), che sono in grado di ricreare la struttura originaria delle directory. L'estrazione dei file dovrebbe avvenire su una directory distinta (ad es. *cpphtp3e_examples*).
- Il CD-ROM contiene *Microsoft Visual C++ 6 Introductory Edition software*: si tratta di un software che consente agli studenti di modificare, compilare ed effettuare il debugging dei programmi in C++. Inoltre il sito web www.deitel.com contiene un breve tutorial su *Visual C++ 6* in formato .PDF consultabile gratuitamente.

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.apogeeonline.com/education/booksite>, eseguendo poi ogni programma via via che questo viene presentato all'interno del testo.

La programmazione orientata agli oggetti

Sin dal primo capitolo cerchiamo di comprendere insieme in che cosa consiste la programmazione orientata agli oggetti, anche se l'analizziamo in maniera formale soltanto a partire dal Capitolo 6. Nei primi cinque capitoli, quindi, introduciamo la programmazione procedurale, ossia analizziamo la componente C del C++, ma iniziamo ad abituare il lettore a "pensare in termini di oggetti" con lo sviluppo progressivo di un caso concreto, presentato nei paragrafi conclusivi di ciascun capitolo. A partire dal capitolo 6 vediamo quali sono le migliorie apportate dal C++ al C e introduciamo la programmazione ad oggetti dal punto di vista formale.

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 – Introduzione: i computer, la programmazione e il C++. Spiega cosa sono i computer, come funzionano e in che modo possono essere programmati. Questo capitolo introduce la programmazione strutturata e spiega perché questo insieme di tecniche abbia favorito una rivoluzione nel modo di scrivere i programmi. Il capitolo traccia brevemente la storia dello sviluppo dei linguaggi di programmazione, dai linguaggi macchina, ai linguaggi assembly e ad alto livello. Si parla anche dell'origine del C++. Il capitolo include un'introduzione agli ambienti di sviluppo tipici del C++, allo sviluppo dei programmi in C++, ai processi decisionali e alle operazioni matematiche. Dopo aver studiato questo capitolo lo studente avrà un'idea su come scrivere programmi in C++ semplici ma funzionanti. In questo capitolo discutiamo anche l'enorme diffusione di Internet e del World Wide Web. Vediamo poi cosa sono gli *spazi dei nomi* e l'istruzione **using**, a beneficio dei lettori che lavorano su compilatori che aderiscono allo standard. Continueremo ad utilizzare i file di intestazione “vecchio stile” nei primi capitoli di questa edizione, mentre utilizzeremo il nuovo stile nel volume Tecniche avanzate, dove analizzeremo diverse novità del linguaggio. Tuttavia ci vorrà ancora qualche anno perché i vecchi compilatori vadano in cantina.. L'approccio alla tecnologia orientata agli oggetti è del tipo full immersion, grazie alle sezioni “Pensare in termini di oggetti” che ne introducono i concetti fondamentali.

Capitolo 2 – Le strutture di controllo. Questo capitolo introduce la nozione di algoritmo come procedura per risolvere un problema e spiega l'importanza di utilizzare in modo efficace le strutture di controllo per scrivere programmi comprensibili, facili da testare e da mantenere, e funzionanti anche già dalla prima compilazione. Il capitolo introduce la struttura sequenziale, le strutture di selezione (**if**, **if/else** ed **else**) e le strutture di iterazione (**while**, **do/while** e **for**). Si esamina la struttura di iterazione in dettaglio e si mettono a confronto i cicli controllati da un valore sentinella e quelli controllati da un contatore. Il capitolo illustra poi la tecnica di raffinamento passo passo, che è di importanza critica per la stesura di programmi strutturati, e presenta lo pseudocodice, uno strumento largamente utilizzato per semplificare la progettazione di un programma. I metodi e gli approcci che utilizziamo nel Capitolo 2 si possono applicare alle strutture di controllo di qualsiasi altro linguaggio, non solo del C++. Questo capitolo aiuta lo studente a sviluppare delle buone abitudini di programmazione, preparandolo ad affrontare le complesse sfide che presentano gli aspetti più avanzati del linguaggio. Il capitolo si conclude con la discussione degli operatori logici **&&** (and), **||** (or) e **!** (not). Introduciamo poi il nuovo operatore del linguaggio **static_cast**, più sicuro del cast “vecchio stile” ereditato dal C. Vogliamo segnalare il nuovo stile che abbiamo introdotto per i sorgenti in C++, più semplice e comprensibile. Nel capitolo discutiamo infine le nuove regole di visibilità per i contatori nei cicli for. Nella sezione “Pensare in termini di oggetti” iniziamo la prima fase di progettazione orientata agli oggetti (OOD) del simulatore di ascensore identificando le classi che lo implementeranno. Introduciamo inoltre UML, con i casi d'uso, i diagrammi delle classi e degli oggetti e i concetti di associazione, molteplicità, composizione, ruolo e collegamento.

Capitolo 3 – Le funzioni. Il capitolo discute la progettazione e la scrittura dei moduli di un programma. Vi illustriamo le funzioni della libreria standard, le funzioni definite dal programmatore, la ricorsione, la chiamata per valore e la chiamata per riferimento. Le tecniche presentate in questo capitolo sono essenziali per creare programmi strutturati correttamente, e sono ancora più importanti nei programmi destinati a gestire problemi del mon-

do reale. Viene presentata la strategia del “divide et impera” come metodo efficace per risolvere problemi complessi, suddividendoli appunto in componenti più semplici che interagiscono tra di loro. Gli studenti apprezzano in genere l’argomento della generazione dei numeri casuali e della simulazione, così come i giochi di dadi che fanno un utilizzo elegante delle strutture di controllo. Il capitolo fornisce un’introduzione rigorosa alla ricorsione e include una tabella che ricapitola i diversi esempi di ricorsione in questo volume e in “Tecniche avanzate”. Alcuni corsi rimandano la trattazione della ricorsione ai capitoli avanzati, mentre noi crediamo che sia preferibile un approccio graduale che inizi già dai primi capitoli del testo. I 60 esercizi che terminano il capitolo ne includono alcuni classici sulla ricorsione, come la Torre di Hanoi. Questo capitolo illustra anche i cosiddetti “miglioramenti” apportati dal C++ al C, tra cui le funzioni **inline**, i parametri di riferimento, gli argomenti di default, l’operatore unario di risoluzione dello scope, l’overloading delle funzioni e i template di funzione. Alla tabella dei file di intestazione aggiungiamo anche altri file che saranno largamente utilizzati in entrambi i volumi del corso. Vi consigliamo di lavorare all’esercizio 3.54, che applica al programma dei dadi una gestione delle scommesse. Nella sezione “Pensare in termini di oggetti” determiniamo la maggior parte degli attributi necessari alle classi che abbiamo individuato. Continuiamo inoltre a introdurre UML, con i diagrammi di stato e delle attività, insieme con i concetti di evento e azione e la loro correlazione ai suddetti diagrammi.

Capitolo 4 – Gli array. Questo capitolo discute la strutturazione dei dati in array, ovvero in gruppi di dati dello stesso tipo correlati, e presenta numerosi esempi di array ad una e più dimensioni. È opinione diffusa che una buona strutturazione dei dati sia almeno tanto importante quanto l’utilizzo efficace delle strutture di controllo per scrivere programmi ben strutturati. Gli esempi del capitolo illustrano varie manipolazioni di array di uso comune, la creazione di istogrammi, l’ordinamento di dati, il passaggio di array a funzioni e includono un’introduzione al campo dei sondaggi e delle inchieste di mercato (con semplici cenni di statistica). L’ordinamento e la ricerca dei dati sono argomenti fondamentali di questo capitolo: si presenta l’algoritmo di ricerca binaria, che presenta un notevole miglioramento di prestazioni rispetto a quella lineare. I 38 esercizi a fine capitolo includono diversi problemi interessanti e stimolanti, come le tecniche di ordinamento avanzate, la progettazione di un sistema di prenotazione aerea automatizzata, un’introduzione alla “turtle graphics” che deve la fama al linguaggio LOGO, ed infine i problemi del “Giro del cavallo” e delle “Otto regine”, che introducono il concetto di programmazione euristica, largamente impiegata nel campo dell’intelligenza artificiale. Gli esercizi si concludono con 8 problemi di ricorsione tra cui l’ordinamento per selezione, i palindromi, la ricerca lineare, la ricerca binaria, le otto regine, la visualizzazione di un array, la visualizzazione di una stringa al contrario e la ricerca del valore minimo presente in un array. In questo capitolo utilizziamo gli array in stile C che, come vedremo nel Capitolo 5, sono in realtà puntatori a locazioni di memoria consecutive. Arriveremo in seguito a vedere gli array come oggetti a tutti gli effetti: nel Capitolo 8 utilizziamo l’overloading degli operatori su una classe **Array** da cui è possibile creare oggetti array più robusti e facili da programmare rispetto a quelli “vecchio stile” del C. Nel Capitolo 9 del volume Tecniche avanzate introdurremo la classe **vector** della “Libreria standard dei template” (STL), che si utilizza in congiunzione con gli iteratori e gli algoritmi discussi in quel capitolo e che consente di vedere e trattare un array come oggetto a tutti gli effetti. Nella sezione “Pensare in termini di oggetti” determiniamo la maggior parte delle operazioni (comportamenti) delle classi individuate. Introduciamo inoltre i diagrammi di sequenza di UML e il concetto di “messaggio inviato a un oggetto”.

Capitolo 5 – Puntatori e stringhe. Questo capitolo presenta una delle caratteristiche del linguaggio più potenti ma anche più difficili da padroneggiare, i puntatori. V'è una spiegazione dettagliata degli operatori di manipolazione dei puntatori, della chiamata per riferimento, delle espressioni di puntamento e dei puntatori a funzioni. C'è una stretta correlazione in C++ tra puntatori, array e stringhe, per cui introduciamo anche i concetti di base sulla manipolazione delle stringhe e illustriamo come funzionano alcune funzioni di manipolazione molto utilizzate, come **getline** (input di una linea di testo), **strcpy** e **strncpy** (copia di una stringa), **strcat** e **strncat** (concatenamento di due stringhe), **strcmp** e **strncmp** (confronto di due stringhe), **strtok** (estrazione di elementi, o token, da una stringa) e **strlen** (calcolo della lunghezza di una stringa). I 49 esercizi a fine capitolo includono la classica simulazione della gara di corsa tra la tartaruga e la lepre, gli algoritmi di mescolamento e distribuzione delle carte da gioco, l'ordinamento veloce (quicksort) ricorsivo e l'attraversamento ricorsivo di un labirinto. Abbiamo incluso anche la sezione speciale "Costruite il vostro computer", che introduce la programmazione in linguaggio macchina e guida il lettore nella progettazione e nell'implementazione di un simulatore di computer, che consente di scrivere ed eseguire dei programmi in linguaggio macchina. Questo esercizio alquanto insolito è una caratteristica del nostro corso e sarà di notevole aiuto ai lettori che vogliono comprendere come funziona realmente un computer. Troviamo che i nostri studenti generalmente si appassionano a questo progetto e spesso ne implementano miglioramenti sostanziali, alcuni dei quali sono suggeriti direttamente da noi nella sezione degli esercizi. Nel Capitolo 4 del volume Tecniche avanzate includeremo un'altra sezione speciale che guida il lettore nella progettazione di un compilatore: il linguaggio macchina prodotto da questo compilatore potrà poi essere eseguito sul simulatore creato nel Capitolo 7: i dati saranno comunicati dal compilatore al simulatore su file sequenziali, come vedremo nel Capitolo 3 del volume Tecniche avanzate. C'è una seconda sezione speciale che include esercizi di manipolazione di stringhe stimolanti come l'analisi dei testi, il word processing, la visualizzazione di dati in diversi formati, il controllo della protezione, la scrittura dell'equivalente in lettere della somma di un assegno, il codice Morse e la conversione metrica dal sistema metrico decimale a quello anglosassone. Il lettore avrà voglia di tornare su questi esercizi dopo aver studiato la classe **string** nel Capitolo 8 del volume Tecniche avanzate. Molti credono che l'argomento dei puntatori sia di gran lunga il più complesso in un corso di programmazione introduttivo. In C e nel C++ non orientato agli oggetti gli array e le stringhe sono in realtà puntatori a celle di memoria consecutive, e sono puntatori persino i nomi delle funzioni. Lo studio attento di questo capitolo dovrebbe ripagarvi di una comprensione profonda di questo complesso argomento. Ricordiamo comunque che ripareremo di array e stringhe come oggetti a tutti gli effetti più in là nel corso. Nel Capitolo 8 utilizziamo l'overloading degli operatori per creare le classi **Array** e **String**. Nel Capitolo 8 del volume Tecniche avanzate discutiamo della classe **string** della Libreria standard e mostriamo come manipolare gli oggetti **string**. Nel Capitolo 9 di tale volume discutiamo anche la classe **vector**. Nella sezione "Pensare in termini di oggetti" determiniamo la maggior parte delle collaborazioni (interazioni tra oggetti) che servono a implementare il sistema, rappresentandole anche con i diagrammi delle collaborazioni UML. A fine sezione abbiamo inserito una bibliografia e un elenco di risorse in rete su UML, che mettono a disposizione tra le altre cose le specifiche di UML 1.3 e materiali come tutorial, FAQ, articoli e software.

Capitolo 6 – Le classi e l'astrazione dei dati. Questo capitolo inizia la studio della programmazione orientata agli oggetti. Questo corso è una splendida opportunità di insegnare l'astrazione dei dati nel modo che riteniamo più giusto, ovvero attraverso un linguaggio, il

C++, che è stato pensato espressamente per implementare tipi di dati astratti (ADT). Negli ultimi anni l'astrazione dei dati è diventata uno degli argomenti più importanti dei corsi introduttivi di programmazione. I capitoli 6, 7 e 8 ne includono una discussione rigorosa. Il Capitolo 6 mostra l'implementazione degli ADT come **struct**, la loro implementazione come classi in stile C++ e le ragioni per cui quest'ultima è superiore alla prima, l'accesso ai membri di una classe, la separazione di interfaccia e implementazione, l'utilizzo di funzioni di accesso e di utilità, l'inizializzazione di oggetti tramite costruttori, la loro distruzione tramite distruttori, l'assegnamento di default membro a membro e il riutilizzo del software. Gli esercizi a fine capitolo stimolano gli studenti a sviluppare classi per numeri complessi, numeri razionali, orari, date, rettangoli, interi di grandi dimensioni e gioco del tris. Generalmente gli studenti si appassionano ai programmi di gioco. Il lettore più incline alla matematica gradirà gli esercizi sulla classe **Complex** (numeri complessi), sulla classe **Rational** (numeri razionali) e sulla classe **HugeInteger** (numeri di qualsiasi dimensione). Nella sezione "Pensare in termini di oggetti" utilizziamo il diagramma delle classi UML sviluppato nelle sezioni precedenti per determinare i file di intestazione C++ che definiranno le classi. Introduciamo inoltre il concetto di handle a un oggetto e ne iniziamo a studiare l'implementazione in C++.

Capitolo 7 - Le classi: seconda parte. Questo capitolo continua lo studio delle classi e dell'astrazione dei dati. Il capitolo spiega come dichiarare e utilizzare oggetti e funzioni membro costanti, la composizione che permette di creare nuove classi che contengono come membri oggetti di altre classi, le funzioni e le classi **friend** che hanno diritti di accesso speciali ai dati **private** e **protected** delle classi, il puntatore **this** che consente ad un oggetto di conoscere il proprio indirizzo, l'allocazione dinamica della memoria, i membri **static** che consentono di memorizzare e manipolare dati comuni a tutti gli oggetti di una classe, e mostra infine alcuni esempi di tipi di dati astratti molto utilizzati (array, stringhe e code), di classi container e di iteratori. Gli esercizi del capitolo chiedono allo studente di sviluppare una classe "conto corrente" e una classe per insiemi di interi. Nella nostra discussione degli oggetti costanti facciamo brevemente riferimento alla nuova parola chiave **mutable** che, come vedremo nel Capitolo 10 del volume "Tecniche avanzate", serve per modificare l'implementazione "invisibile" degli oggetti **const**. Nel capitolo discutiamo anche l'allocazione dinamica della memoria con **new** e **delete**. Se **new** non riesce ad allocare memoria restituisce un puntatore **0** nelle vecchie versioni del C++: noi utilizziamo questo stile in tutto questo volume e nel primo capitolo del volume Tecniche avanzate. Nel Capitolo 2 di tale volume, invece, introduciamo il nuovo comportamento di **new** in caso di errori, cioè il "lancio di un'eccezione". Abbiamo inserito la spiegazione dei membri **static** in un esempio di videogioco. In tutto in libro, così come nei nostri seminari, poniamo l'accento sull'importanza di tenere nascosti i dettagli di implementazione ai client di una classe. Tuttavia i dati **private** sono perfettamente visibili nei file di intestazione della nostra classe, cosa che naturalmente rivela tali dettagli. Abbiamo quindi introdotto una nuova sezione sulle classi proxy, che rappresentano un meccanismo efficace per nascondere l'implementazione ai client di una classe. Nella sezione "Pensare in termini di oggetti" presentiamo il codice C++ completo del programma di simulazione, circa 1000 linee di codice, e lo analizziamo in dettaglio. Il codice deriva direttamente dalla progettazione UML delle sezioni precedenti e tiene conto di tutte le "buone abitudini di programmazione" che predichiamo, tra cui l'utilizzo di dati membro e funzioni **static** e **const**. Parliamo inoltre dell'allocazione dinamica della memoria, della composizione e dell'interazione tra oggetti per mezzo degli handle, ed infine del problema dell'inclusione circolare dei file che si può evitare grazie alle dichiarazioni anticipate.

Capitolo 8 – L’overloading degli operatori. È tra gli argomenti più importanti di un corso di C++ ed è molto apprezzato dagli studenti, perché lo trovano perfettamente conseguente alla discussione degli ADT dei Capitoli 6 e 7. L’overloading degli operatori consente di indicare al compilatore come interpretare gli operatori già esistenti quando vengono utilizzati con i nuovi oggetti creati dall’utente. Il C++ sa già come utilizzare questi operatori con gli oggetti dei tipi predefiniti come interi, numeri a virgola mobile e caratteri. Supponiamo dunque di creare una nuova classe di stringhe: che cosa potrebbe significare il segno più tra due stringhe? Molti programmatori utilizzano il segno più con le stringhe per indicare il concatenamento. Nel Capitolo 8 si spiega come “sovraccaricare” (in inglese *overload*) il segno più in modo tale che quando il compilatore lo rileva in un’espressione tra due oggetti stringa genera una chiamata alla *funzione operatore* che effettua il concatenamento. Il capitolo discute i concetti di base dell’overloading degli operatori, le restrizioni, l’overloading con le funzioni membro di una classe e le funzioni non membro, l’overloading di operatori unari e binari e la conversione tra tipi. Una caratteristica del capitolo è costituita dai diversi esempi significativi tra cui una classe di array, una di stringhe, una di date, una di interi di grandi dimensioni e una di numeri complessi (gli ultimi due compaiono negli esercizi insieme con il codice completo). Lo studente più incline alla matematica vorrà cimentarsi negli esercizi con la creazione di una classe di polinomi. Questo materiale è diverso da quello che normalmente si può trovare in corsi simili al nostro. L’overloading degli operatori è un argomento complesso ma di notevole valore aggiunto, perché consente di dare quella “rifinitura” finale alle proprie classi. Le discussioni sulla classe **Array** e sulla classe **string** sono di particolare interesse per gli studenti che hanno intenzione di studiare successivamente le classi della Libreria standard **string** e **vector**. Grazie alle tecniche descritte nei Capitoli 6, 7 e 8 è possibile creare una classe **Date** che avremmo voluto vedere utilizzata già dal 1980, perché ci avrebbe risparmiato gran parte dei problemi che abbiamo incontrato nel passaggio all’anno 2000. Gli esercizi incoraggiano lo studente a integrare l’overloading degli operatori nelle classi **Complex**, **Rational** e **HugeInteger**, per poter manipolare questi oggetti con dei simboli, come in matematica, piuttosto che con chiamate di funzione, come accadeva negli esercizi del Capitolo 7.

Capitolo 9 – L’ereditarietà. Questo capitolo parla di una delle funzionalità principali dei linguaggi orientati agli oggetti. L’ereditarietà è una forma di riutilizzo del software grazie alla quale è possibile sviluppare nuove classi in modo facile e rapido “assorbendo” le funzionalità di altre classi esistenti e aggiungendone di nuove. Il capitolo discute i seguenti concetti: classe base e classe derivata, membri **protected**, ereditarietà di tipo **public**, **protected** e **private**, classi base dirette e indirette, costruttori e distruttori di classe base e derivata; infine discute dell’ingegneria del software che utilizza l’ereditarietà. Il capitolo confronta l’ereditarietà (relazione “is a”) con la composizione (relazione “has a”) e introduce le relazioni “uses a” e “knows a”. Una caratteristica di questo capitolo sta nei casi di studio significativi proposti. In particolare, un caso alquanto lungo implementa la gerarchia di classi punto/cerchio/cilindro. Il capitolo si conclude con un caso di ereditarietà multipla, una caratteristica avanzata del C++ che consente di derivare gli attributi e i comportamenti di una classe da diverse classi base. Gli esercizi chiedono allo studente di confrontare la creazione di nuove classi con l’ereditarietà e con la composizione, di estendere le diverse gerarchie di ereditarietà discusse nel capitolo, di scrivere una gerarchia per i quadrilateri, i trapezi, i parallelogrammi, i rettangoli e i quadrati ed infine di creare una gerarchia più generale di forme geometriche che distingue tra forme nel piano e forme nello spazio. Abbiamo modificato la gerarchia dell’ereditarietà relativa ai membri della

comunità universitaria per fornire un esempio di ereditarietà multipla. Nel Capitolo 10 del volume Tecniche avanzate continueremo la discussione sull'ereditarietà multipla esponendo i problemi causati dalla cosiddetta "ereditarietà romboidale" e mostrando come si possono risolvere grazie alle classi base virtuali. Nella sezione "Pensare in termini di oggetti" introduciamo il concetto di ereditarietà nel programma presentato nel Capitolo 7. Sugeriamo anche ulteriori miglioramenti al codice, ma li lasciamo agli studenti come esercizio, perché possano cimentarsi con gli strumenti di progettazione con cui avranno familiarizzato nelle sezioni precedenti.

Capitolo 10 – Le funzioni virtuali e il polimorfismo. Questo capitolo introduce un'altra funzionalità importante della programmazione orientata agli oggetti, il comportamento polimorfico. Quando diverse classi sono correlate per mezzo dell'ereditarietà a una classe base comune, ogni oggetto di una classe derivata può essere trattato come un oggetto di classe base: ciò consente di scrivere programmi generici e indipendenti dai tipi specifici delle classi derivate. Si possono gestire con uno stesso programma nuovi tipi di oggetto, e ciò contribuisce a rendere un sistema estensibile. Il polimorfismo consente di evitare la complessa logica degli **switch** nei programmi, a favore di una logica lineare più comprensibile. Lo screen manager di un videogioco, per esempio, può inviare semplicemente un messaggio "disegna" a tutti gli elementi di una lista concatenata di oggetti da disegnare sullo schermo. Ciascun oggetto saprà come disegnare se stesso. Inoltre si può aggiungere un nuovo oggetto al programma senza modificare quest'ultimo, sempre che il nuovo oggetto sappia come disegnare se stesso. Questo stile di programmazione si utilizza tipicamente per implementare le comuni interfacce grafiche per gli utenti (dette GUI). Il capitolo discute come si implementa un comportamento polimorfico con le funzioni virtuali. Si mostra la distinzione tra classi astratte, da cui non è possibile istanziare oggetti, e classi concrete, da cui ciò è possibile. Le classi astratte sono utili per fornire un'interfaccia ereditabile a classi che non sono in una gerarchia. Il capitolo presenta due casi di studio importanti sul polimorfismo: un libro paga elettronico e una seconda versione della gerarchia di forme geometriche discussa nel Capitolo 9. Gli esercizi chiedono allo studente di discutere diversi approcci e questioni concettuali, di includere classi astratte alla gerarchia di forme geometriche, di sviluppare un semplice pacchetto grafico e di modificare la classe **employee** introdotta nel capitolo. I due casi di polimorfismo mostrano una differenza di stili nell'ereditarietà: il libro paga fa un uso chiaro e "sensibile" dell'ereditarietà, mentre il secondo esempio è un esempio di "ereditarietà strutturale", che non è naturale come il primo tipo ma ugualmente corretta da un punto di vista formale e funzionale. Abbiamo deciso di includere questo secondo esempio in relazione alla sezione "L'implementazione di polimorfismo, funzioni virtuali e binding dinamico". I nostri seminari si rivolgono a programmatori esperti ed essi apprezzavano i nostri esempi di polimorfismo, ma dicevano che alla nostra presentazione mancava qualcosa: è vero, mostravamo come programmare in C++ con il polimorfismo, ma essi volevano qualcosa in più. Nello specifico ci dissero che erano preoccupati del consumo aggiuntivo di risorse che il polimorfismo inevitabilmente implicava, per cui ci chiesero di fornire loro una spiegazione più approfondita del modo in cui è implementato in C++, e di conseguenza di quanto "costa" in termini di tempo di esecuzione e di memoria. Abbiamo dato ascolto a questa loro richiesta sviluppando una sezione che illustra le *vtable* (tabelle delle funzioni virtuali) create automaticamente dal compilatore quando incontra il polimorfismo. Abbiamo applicato queste tabelle alle classi della gerarchia di forme geometriche. Abbiamo avuto conferma dal nostro pubblico che esse davano loro le informazioni necessarie per decidere caso per caso se adotta-

re uno stile di programmazione polimorfico. Questa presentazione si trova nella Sezione 10.10 e l'illustrazione delle vtable è in Figura 10.2.

Capitolo 11 – Gli stream di input/output del C++. Questo capitolo contiene una discussione esauriente del nuovo input/output orientato agli oggetti introdotto in C++. Illustriamo le varie funzionalità di I/O del C++ tra cui l'output con l'operatore di inserimento nello stream, l'input con l'operatore di estrazione dallo stream, l'I/O orientato ai tipi di dato (un miglioramento significativo rispetto al C), l'I/O formattato e non formattato (per migliori prestazioni), i manipolatori degli stream per il controllo della base numerica di uno stream (decimale, ottale o esadecimale), l'I/O dei numeri a virgola mobile, il controllo dell'ampiezza del campo, i manipolatori definiti dall'utente, gli stati di formattazione di uno stream, gli stati di errore di uno stream, l'I/O di oggetti di tipi definiti dall'utente e il collegamento di uno stream di output a uno stream di input per assicurare che la richiesta di un input all'utente appaia effettivamente prima che il sistema si metta in attesa. L'insieme di esercizi chiede allo studente di scrivere diversi programmi che testano la maggior parte delle funzionalità di I/O discusse nel corpo del capitolo.

