

Indice

Prefazione XVII

xi	Capitolo 1	Introduzione ai linguaggi orientati agli oggetti
	1.1	Prerequisiti 2 <i>Terminologia 2</i>
	1.2	Perché usare un linguaggio orientato agli oggetti? 3 <i>Terminologia dei linguaggi orientati agli oggetti 3</i>
	1.3	Storia 3 <i>Linguaggi procedurali 4</i> <i>La crisi del software 4</i>
	1.4	Supporto alla strutturazione e al test dei programmi 6
	x 1.5	Il processo di traduzione del linguaggio 7 <i>Interpreti 7</i> <i>Compilatori 7</i> <i>Il processo di compilazione 8</i>
	x 1.6	Utilità degli oggetti 9 <i>Creazione di entità multiple 10</i> <i>Astrazione dei dati 10</i> <i>Organizzazione del codice 11</i>
	x 1.7	Ereditarietà 12 <i>Sforzo di conservazione 12</i> <i>Creazione di programmi estensibili 12</i> <i>Overloading 14</i>
	x 1.8	Passaggio alla programmazione a oggetti 14 <i>Apprendimento del C++ 15</i>
	x 1.9	Caratteristiche di un linguaggio orientato agli oggetti 16
	x 1.10	Sommario: le attività della programmazione orientata agli oggetti 17

VI Indice

- Come il C++ supporta la programmazione orientata agli oggetti* 17
- × 1.11 Note riguardanti Smalltalk 18
- 1.12 Un C migliore 18
- Si × **Capitolo 2 Utilizzo di classi predefinite**
- × 2.1 Strumenti per la compilazione separata 20
 - Dichiarazioni e definizioni* 20
 - Linking* 23
 - Uso delle librerie* 24
- × 2.2 Il primo programma C++ 26
 - Utilizzo delle classi iostream* 26
 - Aspetti fondamentali della struttura di un programma* 26
 - Ciao, mondo!* 27
 - Esecuzione del compilatore* 28
 - Strategie di traduzione del C++* 28
- × 2.3 Make: uno strumento essenziale per la compilazione separata 29
 - Compiti di make* 30
 - File di make utilizzati nel libro* 31
- × 2.4 Ulteriori informazioni sulla classe iostream 31
 - Concatenazione di stringhe* 31
 - Lettura di caratteri in input* 32
 - Semplice manipolazione di file* 32
- × 2.5 Controllo di flusso in C/C++ 34
 - Vero e Falso in C* 34
 - If-else* 34
 - While* 35
 - Do-while* 36
 - For* 36
 - Le parole chiave break e continue* 37
 - Switch* 39
- × 2.6 Introduzione agli operatori del C e del C++ 40
 - Precedenza* 40
 - Autoincremento e autodecremento* 41
- × 2.7 Uso dello Standard I/O per una semplice gestione di file 42
 - Programma "cat"* 42
 - Gestione di buffer di input* 43
- × 2.8 Programmi di utilità che usano iostream e Standard I/O 45
 - Pipe* 45
 - Programma di analisi di un testo* 45
 - Sostituzione di caratteri di tabulazione* 47
 - Gestione di file mediante la classe iostream* 47

Si × Capitolo

- ✕ 2.9 Notazione per i commenti 48
- ✕ 2.10 Makefile per gli esempi del capitolo 49
- ✕ 2.11 Sommario 50
- 51 ✕ **Capitolo 3 Creazione di classi in C++**
- ✕ 3.1 Introduzione ai tipi di dati in C++ 52
 - Tipi di dati predefiniti di base* 52
 - Qualificatori* 53
- ✓ 3.2 Visibilità delle variabili 54
 - Definizione di variabili* 55
- ✕ 3.3 Allocazione di memoria 56
 - Variabili globali* 56
 - Variabili locali* 57
 - static* 57
 - extern* 58
 - Costanti* 60
 - volatile* 62
- ✕ 3.4 Operatori e loro uso in C/C++ 62
 - Assegnamento* 63
 - Operatori matematici* 63
 - Operatori relazionali* 65
 - Operatori logici* 65
 - Operatori di manipolazione di bit* 65
 - Operatori di scorrimento* 66
 - Operatori unari* 68
 - Operatore condizionale o operatore ternario (? :)* 69
 - L'operatore virgola* 69
 - Errori comuni nell'uso degli operatori* 70
 - Operatori di conversione di tipo* 70
 - Operatore sizeof()* 71
- ✓ 3.5 Creazione di funzioni in C e in C++ 71
 - Prototipi di funzione* 71
 - Uso delle funzioni di libreria C* 74
 - Creazione di una propria libreria usando il programma Librarian* 74
- ✓ 3.6 Caratteristiche proprie delle funzioni C++ 75
 - Funzioni inline* 75
 - Overloading delle funzioni in C++* 77
 - Argomenti di default* 78
- ✕ 3.7 Definizione della classe 79
 - Riflessioni sugli oggetti* 80
 - Dichiarazione e definizione* 82

- Costruttori e distruttori (inizializzazione e distruzione) 82
- ✓ 3.8 File header 87
 - Librerie di funzioni e compilazione separata 88
 - Prevenire la ridichiarazione di classi 89
- ✗ 3.9 Definizione di funzioni membro di una classe 91
 - L'operatore :: 91
 - Chiamate di altre funzioni membro 93
 - Funzioni amiche: accesso agli elementi privati di altre classi 94
- ✗ 3.10 Altri costrutti simili alla classe 99
 - Le strutture: classi in cui tutti gli elementi sono pubblici 99
 - Rendere più chiari i programmi mediante i tipi enumerativi 100
 - Risparmio di memoria con le unioni 101
- ✗ 3.11 Funzioni membro di tipo static 104
- ✗ 3.12 Funzioni membro di tipo const e volatile 105
 - Oggetti di tipo const 105
 - Funzioni membro di tipo const 105
 - Oggetti di tipo volatile e funzioni membro 108
- ✓ 3.13 Consigli per il debugging 108
 - Flag di debugging 110
 - Convertire un nome di variabile in una stringa 111
 - La macro ANSI C assert() 111
 - Tecniche di debugging combinate 112
- ✗ 3.14 Stile di formattazione 113
- 3.15 Makefile per gli esempi del capitolo 114
- ✗ **Capitolo 4 Puntatori e riferimenti**
- ✗ 4.1 Indirizzi come caselle postali 117
- ✗ 4.2 Puntatori 118
 - Uso dei puntatori negli array 118
 - Funzioni che modificano i propri argomenti 119
 - Accesso alla memoria 121
 - Allocazione dinamica della memoria 121
- ✗ 4.3 Uso di puntatori e indirizzi 121
 - Aritmetica dei puntatori 122
- ✗ 4.4 Puntatori a variabili 124
 - Puntatori void 125
- ✗ 4.5 Puntatori ad array 127
 - Numerazione negli array 128
 - Inizializzazione degli aggregati 129
 - Array di puntatori 136

- ✗ 4.6 La dimensione di un puntatore (modelli di memoria) 138
 Puntatori far 139
 Esempi di dimensioni di puntatori 140
- ✗ 4.7 Indirizzi di funzioni 140
 Definizione di un puntatore a funzione 141
 Dichiarazioni e definizioni complesse 141
 Uso di un puntatore a funzione 142
 Array di puntatori a funzioni 143
- ✗ 4.8 Esempi di uso di un puntatore 146
 Statistiche sui caratteri contenuti in un file 146
 Passaggio di argomenti di dimensione sconosciuta 147
 Funzioni che modificano i parametri attuali 148
 Passaggio di strutture e di puntatori a oggetti 150
 Esame del contenuto di un float 152
- ✗ 4.9 Riferimenti 154
 Uso dei riferimenti in C++ 155
 Sintassi dei riferimenti 155
 Perché i riferimenti sono essenziali 157
 Riferimenti indipendenti 159
 Significato del ritorno di un indirizzo 160
 Puntatori e riferimenti a oggetti 164
 Riferimenti membri di una classe 165
- ✗ 4.10 Quando usare i riferimenti 166
 Vantaggi dei riferimenti 166
 Problemi con i riferimenti 166
 Indicazioni per il passaggio di argomenti 167
- ✗ 4.11 Makefile per gli esempi del capitolo 169

SI ✗ Capitolo 5 Overloading delle funzioni e degli operatori

- ✗ 5.1 Sintassi dell'overloading degli operatori 172
- ✗ 5.2 Esempi di overloading degli operatori 173
 Una classe matematica 173
- ✗ 5.3 Overloading complesso degli operatori 184
 Dimensioni multiple con operator[] 186
 Overloading dell'operatore virgola 192
 Puntatori intelligenti 193
 Il costruttore di copia e l'operatore operator=() possono essere creati dal compilatore 194
- ✗ 5.4 Creazione di propri operatori di conversione di tipo 195
 Problemi con troppe conversioni di tipo 198
 Consigli di progettazione per la conversione di tipo 200
- ✗ 5.5 Esempio: personalizzazione delle funzioni iostream 204

- 5.6 Scegliere funzioni amiche o membro per l'overloading degli operatori 207
 - 5.7 Overloading delle funzioni 209
 - Acquisire l'indirizzo di una funzione overloaded* 210
 - Type-Safe Linkage* 211
 - 5.8 Makefile per gli esempi del capitolo 212
- SI ✕ **Capitolo 6 Creazione di oggetti a tempo di esecuzione**
- 6.1 Il pregiudizio di alcuni popolari linguaggi 216
 - 6.2 Creazione dinamica di oggetti 217
 - Lo stack* 217
 - La memoria libera* 218
 - Life: un esempio di simulazione e di creazione di modelli* 220
 - 6.3 Oggetti di dimensione arbitraria 230
 - Un array dinamico* 230
 - Duplicazione di spazio per le stringhe* 237
 - 6.4 Il meccanismo di creazione dinamica degli oggetti 240
 - L'ordine delle chiamate del costruttore e del distruttore* 240
 - Operazioni interne* 245
 - 6.5 Cambiare il comportamento di new e delete 247
 - Esaurimento della memoria libera* 248
 - Overloading classe per classe di new e delete* 249
 - Semplice routine di garbage-collection* 250
 - 6.6 Errori comuni nell'allocazione dinamica della memoria 252
 - Calcolo della dimensione di un oggetto* 252
 - 6.7 Uso dei riferimenti con l'allocazione dinamica della memoria 255
 - 6.8 Posizionamento di oggetti in specifiche locazioni di memoria 256
 - 6.9 Makefile per gli esempi del capitolo 258
- SI ✕ **Capitolo 7 Reimpiego di codice in C++**
- 7.1 Sintassi per la composizione e l'ereditarietà 260
 - Composizione: reimpiego di codice per mezzo di oggetti membri* 260
 - Sintassi per l'ereditarietà* 262
 - Composizione con puntatori a oggetti* 266
 - 7.2 Esempi di composizione ed ereditarietà 267
 - Classe per la gestione degli errori* 267
 - Approccio mediante iostream* 270

220

40

52

- Classe per la gestione di temporizzazioni* 278
 - 7.3 Reimpiego di codice con l'ereditarietà 281
 - Classe per la gestione dello schermo* 281
 - Aggiunta di funzionalità mediante l'ereditarietà* 284
 - Ulteriori miglioramenti per cursor_controller2* 286
 - 7.4 Memorizzazione di oggetti su disco 294
 - Utilizzo di classi persistenti* 295
 - Considerazioni riguardanti il progetto* 296
 - Funzioni iostream per file di I/O* 299
 - Creazione di una classe persistente* 303
 - Suggerimenti per apportare miglioramenti* 305
 - 7.5 Una lista che può salvare e ricaricare se stessa 305
 - Definizione del database* 306
 - Una classe che contiene un record del database* 307
 - Una classe per costruire una lista di db_record* 309
 - Aggiungere record al file di database* 311
 - Ricerca di record nel file di database* 313
 - Conversione da un file ASCII* 315
 - 7.6 Accesso agli elementi della classe base 318
 - Disabilitazione di funzioni membro in una classe derivata* 319
 - 7.7 Ereditarietà multipla 321
 - Sintassi dell'ereditarietà multipla* 322
 - Problemi con l'ereditarietà multipla* 323
 - 7.8 Makefile per gli esempi del capitolo 326
- Capitolo 8 Scrittura di programmi estensibili**
- 8.1 Codifica della gerarchia della classe transportation 333
 - 8.2 Funzioni virtuali 339
 - Analisi delle funzioni virtuali* 345
 - Meccanismo delle funzioni virtuali* 346
 - Regole per le funzioni virtuali* 350
 - Funzioni virtuali ed efficienza* 350
 - 8.3 Un sistema di menu estensibile 350
 - Una classe per rappresentare una singola opzione del menu* 352
 - Una classe per gestire un insieme di opzioni* 354
 - Verifica del sistema di menu* 358
 - 8.4 Classi astratte 366
 - Una classe astratta per il debugging* 366
 - Una classe astratta per il garbage-collection* 369
 - 8.5 Uso delle classi astratte per la simulazione 372
 - 8.6 Costruzione e distruzione 379

- Un problema di fermata* 381
 - Distruttori virtuali puri* 382
- 8.7 Makefile per gli esempi del capitolo 382
- Capitolo 9 Argomenti e valori di ritorno**
- 9.1 Passaggio per nome, per valore, o per riferimento 386
 - Passaggio per riferimento* 386
 - Quando usare il passaggio per valore* 387
- 9.2 Attività nascoste in C 387
 - Passaggio di argomenti in C* 387
 - Ritorno di valori in C* 388
 - Il problema delle strutture* 388
- 9.3 Attività nascoste in C++ 389
 - Pericoli nel ritornare indirizzi* 390
- 9.4 Il costruttore di copia X(X&) 390
 - Copia all'interno o all'esterno delle funzioni* 391
- 9.5 Assegnamento di un oggetto 393
 - Caratteristiche uniche di operator=()* 393
 - Variabili temporanee* 394
 - Analisi di un assegnamento* 395
- 9.6 Un esempio di copia profonda: la classe matrix 395
 - Conteggio dei riferimenti* 396
 - operator=() e selezione di un elemento* 396
 - Moltiplicazione e addizione in matrix* 397
 - Una classe matrix completa* 397
 - Avvertimento* 401
- 9.7 Mantenere traccia della creazione e della distruzione di oggetti 402
 - Implementazione del tracciato per la classe matrix* 405
- 9.8 Ritornare oggetti con *new 411
 - Altre informazioni sui riferimenti* 411
 - Uso corretto di *new* 412
 - Uso improprio di *new* 413
- 9.9 Struttura di un interprete orientato agli oggetti in C++ 415
 - Soluzione di Smalltalk* 415
 - Controllo di tipo* 416
 - Creazione di un oggetto* 417
 - Esempio* 417
 - Il nucleo del progetto* 424
 - Mantenere traccia degli oggetti* 425
 - Creazione di specifici tipi di dati* 425
 - Verifica del progetto* 427

- 9.10 *Costruzione dell'interprete* 431
- 9.10 *Makefile per gli esempi del capitolo* 431
- Capitolo 10 Classi contenitore e modelli in C++**
- 10.1 *Contenitori e iteratori* 434
- 10.1 *Prefisso e postfisso* 438
- 10.2 *Introduzione ai modelli* 438
- 10.2 *Modelli in C++* 440
- 10.3 *Tipi parametrizzati definiti per mezzo del preprocessore* 444
- 10.3 *Istruzioni #pragma* 449
- 10.3 *Funzioni membro di String_stackq* 449
- 10.3 *Parametrizzazione di stackq* 451
- 10.3 *Macro generiche per i tipi parametrizzati* 454
- 10.4 *Argomenti dei modelli* 459
- 10.4 *Costanti all'interno delle classi* 461
- 10.4 *Il modello queue con un argomento size* 461
- 10.5 *Modelli e hardware* 462
- 10.6 *Membri di tipo static nei modelli* 463
- 10.7 *Metodo di generazione delle funzioni modello* 464
- 10.8 *Modelli di funzioni* 473
- 10.9 *Modelli ed ereditarietà* 483
- 10.10 *Uso dei modelli nelle librerie* 486
- 10.11 *Nuova versione del programma Brackets.cpp* 490
- 10.11 *Un editor di tipo batch* 494
- 10.12 *Potenza dei modelli* 497
- 10.13 *Makefile per gli esempi del capitolo* 497
- Capitolo 11 Esempi completi**
- 11.1 *Programmi di utilità riguardanti gli iostream* 500
- 11.1 *Manipolazione di file per mezzo di iostream* 500
- 11.1 *Un "Alias" per MS-DOS* 501
- 11.1 *Marcare le parole in un file* 505
- 11.1 *Uno strumento per controllare la lunghezza di una linea* 507
- 11.1 *Un programma per stampare un file di testo* 509
- 11.2 *Elaborazione della linea di comando* 512
- 11.2 *Flag della linea di comando* 513
- 11.2 *Un semplice programma di utilità* 517
- 11.3 *TAWK: un semplice interprete di database* 518

- Algoritmo a discesa ricorsiva* 519
- La classe field* 520
- Classe csascii* 523
- Verifica delle classi field e csascii* 526
- TAWK: Tiny AWK* 527
- Classe token* 537
- Classe parse_array* 538
- Esecuzione di un file di comandi di TAWK* 538
- Esempio di un file di comandi di TAWK* 539

- 11.4 *Un sistema di controllo basato sul clock* 541
 - Il programma CONTROLR* 541
 - Uso del programma CONTROLR* 543
 - Controllo di eventi orientato agli oggetti* 544
 - Una classe per gestire la temporizzazione* 544
 - Una lista collegata orientata al C++* 549
 - Analisi sintattica del file di comandi* 552
 - Aggiungere un evento alla lista* 562
 - Gestione della lista* 562
 - Indicazioni di progetto* 563
 - Ripartenza del sistema* 563
 - Aumento della velocità di ripartenza* 563
 - Vantaggi di un progetto orientato agli oggetti* 564

- 11.5 *Serpenti sullo schermo* 564
 - Segmenti* 566
 - Costruttori per segment* 567
 - Funzioni membro ricorsive* 571
 - La classe snake* 571
 - Una funzione main() flessibile* 572

- 11.6 *Progetti che utilizzano forme grafiche* 573
 - Incapsulamento dell'interfaccia Grafica Borland* 573
 - Una classe shape generica* 575
 - Implementazione di shape* 577
 - Specifici tipi di shape* 579
 - Poligoni* 581
 - Classi contenitore* 585
 - L'editor di forme grafiche SHED* 585
 - Codici di scansione della tastiera* 589
 - Estensione del programma* 591
 - Generazione casuale di forme grafiche* 593
 - Simulazione con le forme grafiche* 598

- 11.7 *Makefile per gli esempi del capitolo* 602

Appendice A Gestione delle eccezioni

- A.1 Eccezioni in C++ 606
Sintassi 607

Appendice B La Classe matrix

- B.1 File standard per la matrice 610
- B.2 Miglioramenti per incrementare la velocità 611
- B.3 Codice per la classe matrix 611
Dichiarazione di matrix 611
Funzioni di matrix 613
Verifica della classe matrix 625
- B.4 Makefile per la classe matrix 626

Indice analitico