C++

Vol 2

Scott Meyers

ADDISON-WESLEY

IT Press

C++ Vol 2

Kirjoittaja	Scott Meyers
Kääntäjä	Arto Kuvaja
Kansi	Frank Chaumont
Kustantaja	Oy Edita Ab IT Press PL 760 00043 EDITA
	Sähköpostiosoite palvelu@itpress.fi Internet www.itpress.fi
Painopaikka	Oy Edita Ab, Helsinki 2000

Alkuperäisen teoksen nimi on *Effective C++ Second Edition*

ISBN 951-826-192-X

Arvosteluja tästä kirjasta

"Meyersin kirja on todellakin ylistyksen arvoinen... kirja sisältää myös erinomaisen ruuvit ja pultit -oppaan muistinhallinnan rakenteisiin, ja selvityksen C++-kielen periytyvyyden eri tyyppien tarkoituksesta."
-New York Computerist

"Tämä kirja pitää ehdottomasti lukea ennen kuin aloitat ensimmäisen todellisen C++-projektisi ja tämä kannattaa lukea uudelleen kun saat kokemusta."
-comp.lang.c++

"Alaotsikkonaan '50 erityistä tapaa kehittää ohjelmiasi ja työtapojasi' kirjoittaja ei pelkästään tarjoa selviä ohjeita, joita noudattaa kirjoitettaessa C++-kielistä koodia, vaan tarjoaa myös perustelut ja esimerkkejä, jotka kuvaavat niiden käyttöä."

"Suosittelen koko sydämellä C++-ohjelmoinnin tehostaminen -kirjaa kaikille, jotka pyrkivät C++-kielen hallintaan keskitasolla tai ylempänä"
-The C user's Journal

"Tämä on yksi parhaimmista näkemistäni kirjoista, jotka on tarkoitettu keskitason ohjelmoijalle. Se on muotoiltu 'essee-joukkona' käytännön ongelmista, joita C++-ohjelmoija kohtaa... Se on siitä harvinainen ohjelmointikirja, että se on sekä hauska että käytännöllinen."
-comp.lang.c++

"Tuloksena on pieni kirja, joka vastaa laajuudeltaan ja olemukseltaan toista kirjaa, The Elements of Style (William Strunk ja E.B. White). Ainakin minun kirjahyllyssäni nämä kaksi kirjaa eivät ole kaukana toisistaan... Tämä on vaatimaton pieni kirja, joka asettaa tavoitteet selkeästi ja saavuttaa ne."
-C++ Report

Arvosteluja tästä kirjasta

"Tämä kirja sisältää käytännön ohjeita C++-kielen hyväksikäyttöön."
-DEC Professional

"Jokaisen C++-ohjelmoijan pitäisi ei pelkästään omistaa tämä kirja, vaan myös opiskella ja ottaa se käyttöön. Tämän tekstin käyttö on helppoa ja se on hyvin ristiviitattu ja indeksoitu."
-Computer Language

"Tämä on 193-sivuinen mestariteos, jonka luin yhdeltä istumalta... takaan, että ainakin osa näistä viidestäkymmenestä kohdasta edistää ohjelmointiasi ja maksaa takaisin vaatimattoman sijoituksesi... Tämä on hyvin kirjoitettu rehellinen kirja, joka on tarkoitettu C++-ohjelmoijille, jotka tarvitsevat sujuvuutta ja tehokkuutta."
-Stan Kelley-Bootle, UNIX Review

Sisältö

Esipuhe

Tämä kirja on suoraa seurausta kokemuksistani C++-kielen opettajana ohjelmoinnin ammattilaisille. Olen huomannut, että useimmat opiskelijat ovat jo viikon tehokkaan opiskelun jälkeen tottuneet kielen perusrakenteisiin, mutta he tuntuvat olevan vähemmän varmoja kyvyistään yhdistää rakenteiden osat tehokkaalla tavalla. Täten käynnistyi minun yritykseni laatia lyhyet, spesifistiset, helposti muistettavat ohjeistot tehokkaaseen ohjelmistokehitykseen C++-kielellä: yhteenveto asioista, joita kokeneet C++-ohjelmoijat melkein aina tekevät tai välttävät tekemästä.

Olin alunperin kiinnostunut säännöistä, jotka eräänlainen `lint`-tyyppinen ohjelma voisi pakottaa. Johdin siihen asti tutkimustyötä, jonka tarkoituksena oli kehittää työkaluja, joilla voitaisiin tutkia C++-lähdekoodia etsimällä käyttäjien määrittelemien ehtojen väärinkäyttöä.† Valitettavasti tutkimustyö loppui ennen kuin kaupallinen prototyyppi pystyttiin kehittämään. Onneksi nykyään on saatavilla monia kaupallisia, C++-kieltä tarkistavia tuotteita.

Vaikka alkuperäinen kiinnostukseni oli ohjelmoinnin säännöissä, jotka voitaisiin pakottaa automaattisesti, tajusin nopeasti tämän työtavan rajallisuuden. Hyvien C++-ohjelmoijien käyttämien ohjeistojen enemmistöä on liian vaikeaa formalisoida tai on olemassa liian paljon tärkeitä poikkeuksia, jotta ne voitaisiin pakottaa ohjelmaan. Minut johdatettiin täten käsitteeseen, joka on vähemmän tarkka kuin tietokoneohjelma, mutta silti keskitetympi ja osuvampi kuin yleinen C++-oppikirja. Pidät tulosta juuri käsissäsi: kirjaa, joka sisältää 50 erityistä ehdotusta siitä, kuinka voit kehittää C++-ohjelmiasi ja työskentelymallejasi.

Löydät tästä kirjasta ohjeita, mitä sinun pitäisi tehdä ja miksi, ja mitä sinun ei pitäisi tehdä ja miksi ei. Pohjimmiltaan miksi-kysymykset ovat tärkeämpiä kuin mitä-kysymykset, mutta on paljon mukavampaa viitata ohjeistojen luetteloon kuin muistaa yksi tai useampi oppikirja.

---

† Löydät yleisesityksen tutkimuksesta alkuperäisen *Effective C++*-kirjan Web-sivulta osoitteesta: `http://www.awl.com/cp/ec++.html`.

Esitystapani ei ole useimmista C++-kirjoista poiketen järjestetty kielen tiettyjen piirteiden ympärille. Tämä tarkoittaa sitä, että en puhu muodostinfunktioista yhdessä paikassa, virtuaalista muodostinfunktioista toisessa paikassa, periytyvyydestä kolmannessa ja niin edelleen. Jokainen käsitelty asia on sen sijaan räätälöity käsittelyn alla olevaan ohjeistoon, ja ne ohjelmointikielen tietyn piirteen eri näkökulmat, jotka käyn läpi, voivat olla hajotettuna ympäri kirjaa.

Tämän lähestymistavan etu on, että se heijastaa paremmin niiden ohjelmistojärjestelmien monimutkaisuutta, joihin C++-kieli usein valitaan. Nämä ovat järjestelmiä, joissa ohjelmointikielen yksittäisen piirteen ymmärtäminen ei riitä. Kokeneet C++-kehittäjät tietävät esimerkiksi, että avointen funktioiden ja virtuaalisten tuhoajafunktio-muuttujien ymmärtäminen ei välttämättä tarkoita sitä, että ymmärtää sisäisiä virtuaalisia tuhoajafunktio-muuttujia. Nämä taisteluissa haavoittuneet kehittäjät tunnistavat, että C++-kielen eri ominaisuuksien *vuorovaikutusten* ymmärtäminen on kaikkein tärkeintä, jotta kieltä käytettäisiin tehokkaasti. Tämän kirjan järjestys heijastaa tätä perusluontoista totuutta.

Tämän työtavan haitta on se, että sinun pitää ehkä tutkia useammasta kuin yhdestä paikasta, että löytäisit kaikki asiat, joita minulla on kerrottavana C++-kielen tietystä rakenteesta. Jotta saisin minimoitua tämän lähestymisen haitat, olen vapaasti sirotellut ristiviittauksia koko kirjaan, ja kirjan lopussa on täydellinen hakemisto.

Kun valmistelin tätä toista painosta, kunnianhimoani parantaa kirjaa on laimentanut pelko. Kymmenettuhannet ohjelmoijat ovat ottaneet omakseen tämän kirjan ensimmäisen painoksen, enkä halunnut tuhota niitä ominaispiirteitä, jotka viehättivät kirjan ensimmäisessä painoksessa. Sen kuuden vuoden aikana, kun kirjoitin kirjan, C++-kieli on kuitenkin muuttunut, C++-kirjasto on muuttunut (katso Kohta 49), C++-ymmärtämykseni on muuttunut ja C++-kielen tunnustettu käyttö on muuttunut. Paljon asioita on muuttunut ja minusta on tärkeää, että tämän kirjan tekninen materiaali on uusittu niin, että se heijastaa nämä muutokset. Olen tehnyt sen, mitä olen voinut, päivittämällä yksittäisiä sivuja painosten välillä, mutta kirjat ja ohjelmat ovat pelottavan samanlaisia - tulee aika, jolloin paikalliset laajennukset eivät riitä, ja ainoa korvaus on järjestelmänlaajuinen uudelleenkirjoitus. Tämä kirja on tuloksena tästä uudelleen kirjoittamisesta: C++-ohjelmoinnin tehostaminen.

Niille, joille ensimmäinen painos on tuttu, voi olla kiinnostavaa tietää, että kirjan jokainen Kohta on kirjoitettu uudelleen. Uskon, että kirjan yleisrakenne on kuitenkin säilynyt hyvin perusteltuna, joten siihen ei ole tullut paljon muutoksia. Olen säilyttänyt viidestäkymmenestä alkuperäisestä Kohdasta 48 kohtaa, olen yrittänyt korjata muutaman Kohdan otsikon sanamuotoa (sen lisäksi, että olen muokannut mukana seuraavia keskusteluja). Poistetut Kohdat (eli ne, jotka on korvattu täysin uudella materiaalilla) ovat numerot 32 ja 49, vaikka paljon siitä tiedosta, joka oli ennen Kohdassa 32, on jostain syystä löytänyt paikkansa uudelleen muokattuun Kohtaan 1. Olen muuttanut Kohtien 41 ja 42 järjestystä, koska tällä tavalla oli helpompaa esittää niiden sisältämä uudistettu materiaali. Lopuksi, olen muuttanut periytyvyyden nuolten järjestyksen. Ne noudattavat nyt melkein maailmanlaajuista sopimusta, jonka mukaan

periytyneistä luokista osoitetaan perusluokkiin. Tämä on sama sopimus, jota noudatin vuonna 1996 ilmestyneessä kirjassa *More Effective C++*. Tässä kirjassa on luettavissa siitä yleisesitys sivuilla 237 - 238.

Tässä kirjassa olevat ohjeistot ovat kaikkea muuta kuin kattavia, mutta hyvien sääntöjen keksiminen - niiden sääntöjen, jotka ovat käytettävissä lähes kaikissa ohjelmissa koko ajan - on vaikeampaa, kuin miltä se näyttää. Voit tietää muista ohjeistoista muita tapoja, joiden avulla C++-kielellä ohjelmoidaan tehokkaasti. Jos tiedät muita tapoja, olenkin kiinnostunut kuulemaan niistä.

Toisaalta sinusta voi ehkä tuntua, että eräät tämän kirjan Kohdat ovat sopimattomia yleisinä ohjeina; tarkoitan sitä, että on olemassa parempi tapa suorittaa tehtävä, joka on tutkittu kirjassa; yksi tai useampi teknisistä keskusteluista voi olla epäselvä, epätäydellinen tai harhaanjohtava. Rohkaisen sinua kertomaan myös näistä asioista minulle.

Donald Knuthilla on ollut jo monta vuotta tapana antaa pieni palkinto ihmisille, jotka huomauttavat hänen kirjoissaan olevista virheistä. Pääsy virheettömään kirjaan on joka tapauksessa kiitettävä, mutta kun otetaan huomioon ne lukuisat virheitä sisältävät C++-kirjat, jotka on hätiköiden julkaistu markkinoille, minut on suorastaan pakotettu seuraamaan Knuthin esimerkkiä. Niinpä jokaisesta tässä kirjassa olevasta virheestä, joka raportoidaan minulle - on se sitten tekninen, kieliopillinen, kirjapainollinen tai muu - lisään tulevissa painoksissa mielelläni tunnustuksena sen ihmisen nimen, joka ensimmäisenä tuo virheen tietoisuuteni.

Lähetä ehdotuksesi, kommenttisi, arvostelusi ja - (huokaus) - virheilmoituksesi minulle englanninkielisenä osoitteeseen:

Scott Meyers
c/o Publisher, Corporate and Professional Publishing
Addison Wesley Longman, Inc.
1 Jacob Way
Reading, MA 01867
U. S. A.

Voit myös vaihtoehtoisesti lähettää minulle sähköpostia osoitteeseen `ec++@awl.com`.

Suomen kielellä voit lähettää palautetta sähköpostiosoitteeseen palvelu@itpress.fi.

Ylläpidän listaa niistä muutoksista, jotka on tehty tähän kirjaan ensimmäisen painoksen jälkeen, mukaan lukien virheiden korjaukset, selvitykset ja tekniset päivitykset. Tämä lista on saatavilla alkuperäisen *Effective C++* -kirjan kotisivulla osoitteessa `http://www.aw1.com/cp/ec++.html`. Jos haluat saada kopion tästä listasta, mutta sinulla ei ole Internet-yhteyttä, lähetä pyyntö johonkin yllä olevista osoitteista, niin huolehdin siitä, että lista lähetetään sinulle.

SCOTT DOUGLAS MEYERS

STAFFORD, OREGON
JULY 1997

Tunnustukset

Siitä on kulunut lähes kolme vuosikymmentä, kun Kathy Reed opetti minulle, mikä tietokone on ja kuinka sitä ohjelmoidaan, joten luulen, että tämä kaikki on todellakin hänen syytään. Donald French pyysi minua vuonna 1989 kehittämään C++-koulutusmateriaalia Institute for Advanced Professional Studies -yritykseen, joten myös hänen harteilleen kuuluu osa syytä. Stratus Computer -oppilaitoksessa heinäkuun kolmannen päivän viikolla vuonna 1991 olleet luokkani oppilaat eivät olleet ensimmäisiä, jotka ehdottivat, että kirjoittaisin kirjan, jossa olisi yhteenveto väitetyn viisauden helmistä, joita ponnahtaa esiin opettaessani, mutta he olivat ne, jotka lopulta vakuuttivat minut tekemään sen, joten he kantavat osan vastuuta. Olen kiitollinen heille kaikille.

Monilla tämän kirjan Kohdista ja esimerkeistä ei ole erityistä lähdettä, ei ainakaan sellaista, jonka muistaisin. Ne kehittyivät sen sijaan yhdistelmänä kokemuksista C++-kielen käyttäjänä ja opettajana, kollegojeni kokemuksista ja Usenetin C++-kielen uutisryhmien osallistujien ilmaisemista mielipiteistä. Monet esimerkit, jotka nyt ovat standardeja C++-kielen opetuksen yhteisössä - varsinkin merkkijonot - voidaan jäljittää taaksepäin Bjarne Stroustrupin alkuperäiseen painokseen kirjasta *The C++ Programming Language* (Addison-Wesley, 1986). Monet tästä kirjasta löytyvistä Kohdista (esimerkiksi Kohta 17) voidaan myös löytää tuosta alkuperäisestä työstä.

Kohdassa 8 on mukana toteutusidea Steve Clamagen *C++ Report* -lehdessä toukokuussa 1993 olleesta artikkelista, "Implementing `new` and `delete`". Kohdan 9 motivoi kirjassa *The Annotated C++ Reference Material* oleva selostus (katso Kohta 50), ja Kohdat 10 ja 13 ovat John Shewchukin ehdottamia. Kohdassa 10 oleva `operator new` -funktion toteutus perustuu Stroustrupin kirjassa *The C++ Programming Language* (Addison-Wesley, 1991) ja Jim Coplienin kirjassa *Advanced C++: Programming Styles and Idioms* (Addison-Wesley, 1992) oleviin esityksiin. Dietmar Kuhl osoitti Kohdassa 14 kuvaamani määrittelemättömän käyttäytymisen. Doug Lea tarjosi peitenimen käytön esimerkit Kohdan 17 lopussa. Idea käyttää `0L`-tyyppiä `NULL`-vakioon Kohdassa 25 tuli Jack Reevesin maaliskuun 1996 *C++ Report* -lehden artikkelista "Coping with Exceptions". Useat Usenetin eri C++-kielen uutisryhmien jäsenistä auttoivat jalostamaan tuon Kohdan luokan, jolla toteutettiin NULL-pohjaisten osoittimien muunnokset jäsenmallien avulla. Steve Clamagen lähettämä uutis-

ryhmäsanoma lievensi innostustani funktioviittauksiin Kohdassa 28. Kohta 33 sisältää havaintoja seuraavista kirjoista: Tom Cargillin *C++ Programming Style* (Addison-Wesley, 1992), Martin Carrollin ja Margaret Ellisin *Designing ja Coding Reusable C++* (Addison-Wesley, 1995), *Taligent's Guide to Designing Programs* (Addison-Wesley, 1994), Rob Murrayn *C++ Strategies and Tactics* (Addison-Wesley, 1993) samoin kuin tietoa Steve Clamagen julkaisuista ja uutisryhmäsanomista. Kohdan 34 materiaali hyötyi keskusteluistani John Lakosin kanssa, ja kun olin lukenut hänen kirjansa, *Large-Scale C++ Software Design* (Addison-Wesley, 1996). Tuossa Kohdassa oleva kirjekuori/kirje-terminologia tulee Jim Coplienin kirjasta *Advanced C++: Programming Styles and Idiom*s; John Carolan loi ilahduttavan termin, "Cheshire Cat-luokka." Kappaleen 35 Suorakulmio/neliö -esimerkki on otettu Robert Martinin maaliskuun 1999 *C++ Report* -lehden kolumnista "The Liskov Substitution Principle". Mark Lintonin kauan sitten lähettämä `comp.lang.c++`-viesti oikaisi minua ajatuksistani heinäsirkoista ja sirkoista Kohdassa 43. Kohdassa 49 oleva esimerkki perityistä luonteenpiirteistä on otettu Nathan Myersin *C++ Report* -artikkelista heinäkuussa 1995 "A New and Useful Template Technique: Traits" ja Pete Beckerin "C/C++ Q&A" -kolumnista marraskuun 1996 *C/C++ User's Journal* -lehdessä; yhteenvetoni C++-kielen kansainvälistymistuesta perustuu esijulkaisuna olevan kirjan vedokseen (tekijöinä Klaus Kreft ja Angelika Langer). "Hello World" tulee tietenkin Brian Kernighanin ja Dennis Ritchien kirjasta *The C Programming Language* (Prentice-Hall, julkaistu alunperin vuonna 1978).

Monet ensimmäisen painoksen lukijat lähettivät minulle ehdotuksia, joita en pystynyt sisällyttämään kirjan ensimmäiseen versioon, mutta olen hyväksynyt nuo ehdotukset muodossa tai toisessa tähän painokseen. Muut hyötyivät Usenetissä olevista C++-kielen uutisryhmistä lähettämällä oivaltavia huomautuksia tämän kirjan materiaalista. Olen kiitollinen jokaiselle seuraavista yksilöistä, ja olen huomauttanut, jos olen hyötynyt heidän ideoistaan: Mike Kaelbling ja Julio Kuplinsky (Esittely); henkilö, jonka muistiinpanoni tunnistavat nimellä "kaveri Clariksesta"† (Kohta 5); Joel Regen ja Chris Treichel (Kohta 7); Tom Cargill, Larry Gajdos, Doug Morgan ja Uwe Steinmüller (Kohta 10); Roger Scott ja Steve Burkett (Kohta 12); David Papurt (Kohta 13); Alexander Gootman (Kohta 14); David Bern (Kohta 16); Tom Cargill, Tom Chappell, Dan Franklin, ja Jerry Liebelson (Kohta 17); John "Eljay" Love-Jensen (Kohta 19); Eric Nagler (Kappale 22); Roger Eastman, Doug Moore ja Aaron Naiman (Kappale 23); Dat Thuc Nguyen (Kohta 25); Tony Hansen, Natraj Kini ja Roger Scott (Kohta 33); John Harrington, Read Fleming ja Dave Smallberg (Kohta 34); Johan Bengtsson (Kohta 36); Rene Rodoni (Kohta 39); Paul Blankenbaker ja Mark Somer (Kohta 40); Tom Cargill ja John Lakos (Kohta 41); Frieder Knauss ja Roger Scott (Kohta 42); David Braunegg, Steve Clamage ja Dawn Koffman (Kohta 45); Tom Cargill (Kohta

† Huomio tälle kaverille: olin Clariksessa 15. marraskuun viikolla vuonna 1993. Ota yhteys minuun ja tunnistuta itsesi henkilönä, joka korosti sen tärkeyttä mitä muotoa `delete`-operaattorista käytetään `typedef`-komennon kanssa, ja annan sinulle mielelläni tunnustuksen näissä kiitoksissa. Tulen myös jotenkin (*erittäin* pientä - älä innostu liikaa) hyvittämään pateettisen epäonnistumiseni sinun tunnistamisessa.

46); Wesley Munsil (Kappale 47); Randy Mangoba (useimmat luokkamääritykset); ja John "Eljay" Love-Jensen (monet kohdat, joissa käytän double-tyyppiä).

Seuraavat ihmiset arvostelivat ensimmäisen painoksen käsikirjoituksen osittaiset ja/tai täydet vedokset Tom Cargill, Glenn Carroll, Tony Davis, Brian Kernighan, Jak Kirman, Doug Lea, Moises Lejter, Eugene Santos Jr., John Shewchuk, John Stasko, Bjarne Stroustrup, Barbara Tilly ja Nancy L. Urbano. Sain lisäksi seuraavilta raportteja: Nancy L. Urbano, Chris Treichel, David Corbin, Paul Gibson, Steve Vinoski, Tom Cargill, Neil Rhodes, David Bern, Russ Williams, Robert Brazile, Doug Morgan, Uwe Steinmüller, Mark Somer, Doug Moore, Dave Smallberg, Seth Meltzer, Oleg Shteynbuk, David Papurt, Tony Hansen, Peter McCluskey, Stefan Kuhlins, David Braunegg, Paul Chisholm, Adam Zell, Clovis Tondo, Mike Kaelbling, Natraj Kini, Lars Nyman, Greg Lutz, Tim Johnson, John Lakos, Roger Scott, Scott Frohman, Alan Rooks, Robert Poor, Eric Nagler, Antoine Trux, Cade Roux, Chandrika Gokul, Randy Mangoba ja Glenn Teitelbaum. Jokainen näistä ihmisistä oli instrumenttina kehittäessäni kirjaa, jota juuri pidät kädessäsi.

Seuraavat ihmiset arvioivat toisen painoksen vedokset: Derek Bosch, Tim Johnson, Brian Kernighan, Junichi Kimura, Scott Lewandowski, Laura Michaels, Dave Smallberg, Clovis Tondo, Chris Van Wyk ja Oleg Zabluda. Olen kiitollinen kaikille näille ihmisille, mutta varsinkin Tim Johnsonille, jonka yksityiskohtainen arviointi vaikutti lopulliseen käsikirjoitukseen monilla eri tavoilla. Olen myös kiitollinen Jill Huchitalille ja Steve Reissille heidän avustaan hyvien arvostelijoiden etsimisessä, tehtävä, joka on kriittisen tärkeä ja kasvavan vaikea. Dawn Koffman ja Dave Smallberg ehdottivat parannuksia C++-kielen koulutusmateriaaliin, joka on johdettu kirjoistani, ja monet heidän ideoistaan ovat löytäneet tiensä tähän tarkistukseen. Lopuksi, vastaanotin huomautuksia tämän kirjan aikaisempien painosten seuraavilta lukijoilta, ja olen muuttanut tämän nykyisen painoksen ottaen huomioon heidän ehdotuksensa: Daniel Steinberg, Arunprasad Marathe, Doug Stapp, Robert Hall, Cheryl Ferguson, Gary Bartlett, Michael Tamm, Kendall Beaman, Eric Nagler, Max Hailperin, Joe Gottman, Richard Weeks, Valentin Bonnard, Jun He, Tim King, Don Maier, Ted Hill, Mark Harrison, Michael Rubenstein, Mark Rodgers, David Goh, Brenton Cooper, Andy Thomas-Cramer, Antoine Trux, John Wait, Brian Sharon ja Liam Fitzpatrick.

Evi Nemeth (yhteistyössä seuraavien kanssa: Addison-Wesley, the USENIX Association ja The Internet Engineering Task Force) on suostunut varmistamaan sen, että ensimmäisestä painoksesta jäljellejääneet kopiot toimitetaan Itä-Euroopan yliopistojen tietojenkäsittely-laboratorioihin; näiden yliopistojen olisi muuten vaikeaa hankkia tämänkaltaisia kirjoja. Evi suorittaa vapaaehtoisesti tämän palveluksen useille kirjoittajille ja julkaisijoille, ja olen iloinen, että voin auttaa jollain pienellä tavalla. Jos haluaisit lisätietoa tästä ohjelmasta, ota yhteys Eviin osoitteessa `evi@cs.colorado.edu`.

Joskus tuntuu, että julkaisualan näyttelijät vaihtuvat yhtä useasti kuin ohjelmoinnin trendit, joten olen iloinen, että kustannustoimittajani John Wait, markkinointijohtajani Kim Dawley ja tuotantopäällikköni Marty Rabinowitz ovat edelleen samoissa rooleissa, kuin he olivat noina viattomina päivinä vuonna 1991, kun aloitin tämän kirjoittamistyön. Sarah Weaver oli projektipäällikkö tälle kirjalle, Rosemary Simpson antoi apua hakemiston luonnissa, ja Lana Langlois toimi pääasiallisena yhteyshenkilönä ja joka paikan ylikoordinaattorina Addison-Wesleyssä, kunnes hän lähti vih-reämmille - tai ainakin erilaisille - laitumille. Kiitän heitä ja heidän kollegoitaan avusta tuhansissa tehtävissä, jotka erottavat yksinkertaisen kirjoittamisen itse kustannustoiminnasta.

Kathy Wrightillä ei ollut mitään tekemistä tämän kirjan kanssa, mutta hän haluaisi kuitenkin tunnustuksen.

Olen kiitollinen innostuneesta ja herpaantumattomasta kannustuksesta, jonka vaimoni Nancy L. Urbano antoi tälle ensimmäiselle painokselle, sekä minun ja hänen perheelleen. Vaikka kirjan kirjoittaminen oli viimeinen asia, joka minun odotettiin tekevän, ja kirjoittaminen vähensi vapaa-aikaani pelkästään pienestä lähes olemattomaan, he tekivät selväksi, että ponnistus kannatti, jos lopputuloksena oli se, että perheessä on kirjoittaja.

Tuo kirjoittaja on ollut perheessä nyt kuusi vuotta, ja Nancy sieltää silti edelleen tuntejani, kestää teknojaaritteluani ja rohkaisee kirjoittamaan. Hänellä on myös kyky tietää *juuri* se oikea sana, kun en keksi sitä. Nancytön elämä ei ole elämisen arvoista.

Koiramme Persephone ei koskaan anna sekoittaa tärkeysjärjestystä. Aikaraja tai ei, kävelyn aika on aina *nyt*.

Esittely

Ohjelmointikielen perusteiden oppiminen on oma juttunsa; se, että opit suunnittelemaan ja toteuttamaan *tehokkaita* ohjelmia tuolla kielellä, on täysin jotain muuta. Tämä on erityisesti totta C++-kielessä, joka mahtailee epätavallisen laajalla valikoimalla voimaa ja ilmeikkyyttä. Se on rakennettu täysin ominaisuuksin varustetun perinteisen kielen päälle, ja se tarjoaa myös laajan valikoiman olio-ominaisuuksia, samoin kuin tuen malleille ja poikkeuksille.

Kun C++-kieltä käytetään oikein, työskentely on täyttä iloa. Suunnaton kokoelma työtapoja, sekä oliosuuntautuneita että perinteisiä, voidaan ilmaista suoraan ja toteuttaa tehokkaasti. Voit määrittää uusia tietotyyppejä, jotka ovat kaikkea muuta kuin erottumattomia niiden sisäänrakennetuista vastineista, ja silti vankasti joustavimpia. Arvostelukykyisesti valitut ja huolellisesti suunnitellut luokkajoukot - ne, jotka hoitavat muistinhallinnan automaattisesti, peitenimien käytön, alustuksen ja siivoamisen, tyyppien muuntamisen, ja kaikki muut arvoitukset, jotka ovat turmioksi ohjelmistojen kehittäjille - voivat tehdä sovellusten ohjelmoinnista helppoa, intuitiivista, tehokasta ja melkein virheetöntä. Tehokkaiden C++-kielisten ohjelmien kirjoittaminen ei ole kohtuuttoman vaikeaa, jos tiedät kuinka niitä kirjoitetaan.

Silloin kun C++-kieltä käytetään kurittomasti, se voi johtaa koodiin, joka on käsittämätöntä, mahdotonta ylläpitää, laajentamattomissa, aikaansaamatonta ja suorastaan väärää.

Temppu on siinä, että tunnistat nämä C++-kielen näkökulmat, joihin luultavasti tulet kompastumaan, ja opit kuinka niitä vältetään. Siinä on tämän kirjan tarkoitus. Oletan, että tunnet jo C++:n kielenä ja että sinulla on jo hieman kokemusta sen käyttämisestä. Se, mitä tarjoan tässä, on opas siihen, kuinka kieltä käytetään tehokkaasti, jotta ohjelmasi ovat ymmärrettäviä, ylläpidettävissä, laajennettavissa, tehokkaita, ja käyttäytyvät odotetulla tavalla.

Tarjoamani neuvot sijoittuvat kahteen laajaan kategoriaan: yleiset suunnittelustrategiat sekä kielen tiettyjen piirteiden mutterit ja ruuvit.

Keskityn työtapakeskusteluissa siihen, kuinka erilaisten työtapojen välilla valitaan niin, että asiat saadaan loppuunsuoritetuiksi C++-kielellä. Kuinka valitset periytyvyyden ja mallien välillä? Mallien ja yleisten osoittimien välillä? Julkisen ja yksityisen periytyvyyden välillä? Yksityisen periytyvyyden ja kerrosajattelun välillä? Funktioiden kuormittamisen ja parametrien oletusarvojen määrittämisen välillä? Virtuaali- ja ei-virtuaalifunktioiden välillä? Arvoparametrillä välittämisen ja viittausparametrillä välittämisen välillä? Oikea päätöksenteko on tärkeää alusta lähtien, koska väärä valinta voi tulla ilmi vasta paljon myöhemmin kehitysprosessissa, jolloin sen oikaiseminen on usein vaikeaa, aikaa kuluttavaa, rappeuttavaa ja kallista.

Jopa silloin, kun tiedät tarkalleen, mitä haluat tehdä, asioiden oikein tekeminen voi vaatia taitoa. Mikä on oikea palautusarvo sijoitusoperaattorille? Kuinka `operator new` -funktion pitäisi käyttäytyä, kun se ei saa tarpeeksi muistia? Koska muodostinfunktion pitäisi olla virtuaalinen? Kuinka jäsenen alustuslista pitäisi kirjoittaa? Tämänkaltaisten yksityiskohtien selvittäminen on kriittistä, koska epäonnistuminen johtaa melkein aina ohjelman odottamattomaan, mahdollisesti hämäävään käyttäytymiseen. Vielä tähdellisempää on se, että poikkeava käyttäytyminen ei ehkä heti ole ilmeistä, saaden aikaan koodihaamun, joka kulkee laatukontrollin läpi, vaikka se vasta idättelee huomaamattomien virheiden kokoelmaa - tikittäviä aikapommeja, jotka vain odottavat laukeamista.

Tämä ei ole kirja, jonka lukeminen kannesta kanteen olisi järkevää. Sinun ei edes tarvitse lukea sitä alusta loppuun. Materiaali on hajotettu enemmän tai vähemmän itsenäisiin Kohtiin. Kohdat viittaavat kuitenkin säännöllisesti toisiin, joten yksi tapa lukea kirja on aloittaa siitä nimenomaisesta Kohdasta, joka kiinnostaa ja seurata sitten viitteitä sinne minne ne johtavat.

Kohdat on ryhmitelty yleisiin aihealueisiin, joten jos olet kiinnostunut keskusteluista, jotka liittyvät tiettyyn asiaan, kuten muistinhallintaan tai oliosuunnitteluun, voit aloittaa asiaankuuluvasta osasta ja joko lukea suoraan siitä tai aloittaa hyppäämällä siitä. Tulet kuitenkin huomaamaan, että kaikki tässä kirjassa oleva materiaali on aika perustavanlaatuista olevaa tehokkaaseen C++-ohjelmointiin, joten melkein kaikki liittyy lopulta kaikkeen muuhun tavalla tai toisella.

Tämä ei ole C++-kielen hakuteos, eikä myöskään tapa, jolla opit kielen nollasta. Olen esimerkiksi innokas kertomaan sinulle kaikista jipoista, joilla voit kirjoittaa `operator new` -funktiosi (katso Kohdat 7-10), mutta oletan, että voit myös huomata muualla, että tuon funktion täytyy palauttaa `void*`-tyyppinen osoitin ja sen ensimmäisen argumentin täytyy olla tyyppiä `size_t`. On olemassa esittelykirjoja C++-kieleen, jotka sisältävät tämänkaltaista tietoa.

Tämän kirjan tarkoitus on korostaa C++-ohjelmoinnin niitä näkökulmia, joita yleensä käsitellään pintapuolisesti (jos ollenkaan). Muut kirjat kuvaavat kielen eri osia. Tämä kirja kertoo sinulle, kuinka yhdistät nämä osat niin, että lopputuloksena on tehokkaita ohjelmia. Muissa kirjoissa kerrotaan, kuinka saat ohjelmasi kääntymään. Tässä kirjassa kerrotaan sinulle, kuinka vältetään ne ongelmat, joista kääntäjät eivät kerro sinulle.

Kuten useimmat kielet, C++-kielellä on rikas kansanperinne, joka yleensä välittyy ohjelmoijalta ohjelmoijalle osana kielen hienoa suullista perintöä. Tämä kirja on yritykseni kirjata jotain tästä kertyneestä viisaudesta lähestyttävään muotoon.

Tämä kirja rajoittaa samaan aikaan itsensä lailliseen, *siirrettävissä* olevaan C++-kieleen. Kirjassa on käytetty vain kielen ISO/ANSI-standardista löytyviä piirteitä. Tämän kirjan avaimena olevana huolena on siirrettävyys, joten jos olet etsimässä laittomuuksia ja yhteensopimattomuuksia, jotka ovat riippuvaisia toteutuksesta, tämä ei ole paikka löytää niitä.

Standardin kuvaama C++-kieli on valitettavasti joskus erilaista kuin naapurissasi olevien kääntäjäkauppiaiden tukema C++-kieli. Tästä on tuloksena, että kun huomautan paikoista, joissa kielen suhteellisen uudet piirteet olisivat käytännöllisiä, osoitan myös sinulle, kuinka voit tuottaa tehokkaita ohjelmia niiden puuttuessa. Olisi hölmöä ottaa uudet piirteet käyttöön tietämättömänä siitä mitä tulevaisuus varmasti tuo tullessaan, mutta samaan aikaan, et voi pidättää elämääsi siihen asti kunnes uusimmat, suurimmat, kaikkivaltiaat C++-kääntäjät ilmestyvät tietokoneellesi. Sinun täytyy työskennellä saatavilla olevien työkalujen avulla, ja tämä kirja auttaa sinua tekemään niin.

Huomaa, että viittaan kääntäjiin - monikossa. Eri kääntäjät toteuttavat vaihtelevia approksimaatioita standardiin, joten rohkaisen sinua kehittämään koodiasi ainakin kahden kääntäjän alaisuudessa. Tämä auttaa sinua välttämään huomaamatonta riippuvaisuutta kielen laajennuksesta tai sen väärintulkitusta standardista. Se auttaa sinua myös pysymään poissa kääntäjäteknologian laitamilta, toisin sanoen uusista piirteistä, joita tukee vain yksi valmistaja. Tällaiset piirteet ovat usein huonosti toteutettuja (virheellisiä tai hitaita - usein molempia), ja niiden esittelyn yhteydessä C++-kielen yhteisöllä ei ole kokemusta avustaa sinua niiden oikeassa soveltamisessa. Loimuavat polut voivat olla jännittäviä, mutta kun tavoitteesi on tuottaa luotettavaa koodia, on usein paras antaa muiden tehdä metsänraivaus puolestasi.

Se mitä *et* löydä tästä kirjasta, on C++-kielen evankeliumi, Yksi Ainoa Polku täydelliseen C++-ohjelmistoon. Tämän kirjan jokainen Kohta antaa opastusta siihen, kuinka löydät parempia työtapoja, kuinka vältät yleisiä ongelmia tai kuinka saavutat suuremman tehokkuuden, mutta yksikään niistä ei ole yleismaailmallisesti sovellettavissa. Ohjelmiston suunnittelu ja toteutus ovat monimutkaisia tehtäviä, ja muuttumatta laitteiston, käyttöjärjestelmän ja sovelluksen rajoitusten värittämiä, joten parasta mitä voin tehdä, on antaa *ohjeistot* parempien ohjelmien luontiin.

Jos noudatat koko ajan kaikkia ohjeistoja, tulet tuskin lankeamaan kaikkein yleisimpiin ansoihin, jotka ympäröivät C++-kieltä, mutta ohjeistoilla on luonnostaan poikkeuksia. Tämän takia jokaisella Kohdalla on selitys. Selitykset ovat tämän kirjan tärkein osa. Vain silloin, kun ymmärrät jokaisen Kohdan takana olevat järjelliset perustat, voit järkevästi määrittää, onko se sovellettavissa siihen ohjelmistoon, jota olet kehittämässä ja ainutlaatuisiin rajoituksiin, joiden alaisuudessa ahkeroit.

Tämän kirjan tavoite on täten saada oivalluksia siitä, kuinka C++-kieli käyttäytyy, miksi se käyttäytyy sillä tavalla, ja kuinka sen käyttäytymistä käytetään eduksesi. Tämän kirjan Kohtien sokea soveltaminen on selvästi epäasianmukaista, mutta samalla sinun ei tietenkään pitäisi vahingoittaa mitään ohjeistoista ilman, että sinulla on hyvä syy siihen.

Tämän kaltaisessa kirjassa ei ole mitään syytä tulla riippuvaiseksi terminologiasta; tuo urheilumuoto on parasta jättää kielen lakimiehille. On kuitenkin olemassa pieni C++-kielen sanasto, joka jokaisen pitäisi ymmärtää. Seuraavat termit tulevat sen verran usein esiin, että kannattaa varmistaa, että olemme kaikki samaa mieltä siitä, mitä ne tarkoittavat.

Esittely (*declaration*) kertoo kääntäjille olion, funktion, luokan, tai mallin nimen ja tyypin, mutta se jättää pois eräitä yksityiskohtia. Nämä ovat esittelyitä:

```
extern int x;                   // olion esittely

int numDigits(int number);      // funktion esittely

class Clock;                    // luokan esittely

template<class T>
class SmartPointer;             // mallin esittely
```

Määritys (*definition*) toisaalta antaa kääntäjille yksityiskohdat. Määritys on oliolle paikka, jossa kääntäjät varaavat muistia oliolle. Funktiolle tai funktiomallille määritys antaa koodin rungon. Luokalle tai luokanmallille määrityksessä luetellaan luokan tai mallin jäsenet:

```
int x;                          // olion määritys

int numDigits(int number)       // funktion määritys
{                               // (tämä funktio palauttaa
  int digitsSoFar = 1;          // desimaalien määrän
                                // tässä parametrissä)
```

```
  if (number < 0) {
    number = -number;
    ++digitsSoFar;
  }
  while (number /= 10) ++digitsSoFar;
  return digitsSoFar;
}
class Clock {                    // luokan määritys
public:
  Clock();
  ~Clock();
  int hour() const;
  int minute() const;
  int second() const;
  ...
};
template<class T>
class SmartPointer {             // mallin määritys
public:
  SmartPointer(T *p = 0);
  ~SmartPointer();
  T * operator->() const;
  T& operator*() const;
  ...
};
```

Saavumme muodostinfunktioihin. *Oletuksena oleva muodostinfunktio* on se, jota voidaan kutsua ilman mitään argumentteja. Tällaisella muodostinfunktiolla ei ole parametrejä, eikä sillä myöskään ole oletusarvoa jokaiselle parametrille. Jos haluat määrittää olioiden taulukon, tarvitset yleensä oletusmuodostinfunktion:

```
class A {
public:
  A();                           // oletustuhoajafunktio
};
A arrayA[10];                    // 10 muod. funk. kutsutaan

class B {
public:
  B(int x = 0);                  // oletusmuodostinfunktio
};
B arrayB[10];                    // 10 muod.funk. kutsutaan
                                 // jokaista argumentilla 0
```

```
class C {
public:
  C(int x);                    // ei oletusmuod.funktio
};

C arrayC[10];                  // virhe!
```

Huomaat varmaan, että kääntäjäsi hylkivät oliotaulukkoja silloin, kun luokan oletusmuodostinfunktiolla on parametrien oletusarvot. Eräät kääntäjät esimerkiksi kieltäytyvät vastaanottamaan edellämainitun `arrayB`-luokan määrityksen, vaikka sillä onkin C++-kielen standardin siunaus. Tämä on esimerkki eroavuudesta, joka voi olla olemassa C++-kielen standardin kuvauksen ja erityisen kääntäjän toteutuksen välillä. Jokaisella tietämälläni kääntäjällä on jokin näistä vajavuuksista. Siihen asti kunnes kääntäjien kauppiaat tavoittavat standardin, valmistaudu olemaan joustava, ja paistattele siinä kieltämättömässä tosiasiassa, että jonain päivänä ei niin kaukaisessa tulevaisuudessa standardissa kuvattu C++-kieli tulee olemaan sama kuin kieli, jonka C++-kääntäjät ovat hyväksyneet.

Asiasta toiseen, jos haluat luoda oliotaulukon, jossa ei ole olemassa oletusmuodostinfunktiota, tavallinen tapa on määrittää sen sijaan osoittimista koostuva taulukko. Jokainen osoitin voidaan sitten alustaa erikseen käyttämällä operaattoria `new`:

```
C *ptrArray[10];               // muodost.funk. ei kutsuta

ptrArray[0] = new C(22);       // varaa ja muodosta
                               // 1 C-olio

ptrArray[1] = new C(4);        // samat sanat

...
```

Jos palataan terminologia-puolelle, kopiomuodostinta (*copy constructor*) käytetään alustamaan olio eri oliolla, joka on tyypiltään sama:

```
class String {
public:
  String();                    // oletusmuodostinfunktio
  String(const String& rhs);   // kopiomuodostin
  ...

private:
  char *data;
};

String s1;                     // kutsuu oletusmuod.funkt.
String s2(s1);                 // kutsuu kopiomuodostinta
String s3 = s2;                // kutsuu kopiomuodostinta
```

Kopiomuodostimen ehkä tärkein käyttö on määrittää, mitä tarkoittaa välittää ja palauttaa olioita arvoparametrillä. Tutki esimerkkinä seuraavaa (tehotonta) tapaa kirjoittaa funktio, jolla liitetään kaksi `String`-oliota:

```
const String operator+(String s1, String s2)
{
  String temp;

  delete [] temp.data;

  temp.data =
    new char[strlen(s1.data) + strlen(s2.data) + 1];

  strcpy(temp.data, s1.data);
  strcat(temp.data, s2.data);

  return temp;
}

String a("Hello");
String b(" world");
String c = a + b;              // c = String("Hello world")
```

Tämä `operator+`-funktio ottaa kaksi `String`-oliota parametreinä ja palauttaa tuloksena yhden `String`-olion. Sekä parametrit että tulos välitetään arvoparametrillä, joten yhtä kopiomuodostinta kutsutaan alustamaan `s1` `a`:n arvolla, yhtä alustamaan `s2` `b`:n arvolla ja yhtä c `temp`-muuttujan arvolla. Kopiomuodostimeen voi itse asiassa tapahtua joitakin lisäkutsuja, jos kääntäjä päättää luoda väliaikaisia tilapäisolioita, joiden luonti on sallittua. Tärkeä asia tässä on, että arvoparametrillä välittäminen tarkoittaa samaa kuin "kopiomuodostimen kutsumista".

`String`-olioiden `operator+`-funktiota ei itse asiassa toteutettaisi tällä tavalla. `const`-tyyppisen `String`-olion palauttaminen on oikein (katso Kohdat 21 ja 23), mutta kahta parametriä ei välitettäisi viittausparametrinä (katso Kohta 22).

`String`-tyypeille ei itse asiassa kirjoittaisi `operator+`-funktiota ollenkaan, jos pystyisit välttämään sitä, mutta sinun pitäisi pystyä välttämään sitä melkein aina. Tämä johtuu siitä, että perus-C++-kirjasto (katso Kohta 49) sisältää merkkijonotyypin (ovelasti nimetty nimellä `string`), samoin kuin `operator+`-funktion `string`-olio, joka tekee melkein saman kuin mitä edellä mainittu `operator+` tekee. Käytän tässä kirjassa sekä `String`- että `string` -olioita, mutta käytän niitä erilaisilla tavoilla. (Huomaa, että aiempi on isolla alkukirjaimella, jälkimmäinen ei ole.) Jos tarvitsen vain yleisen merkkijonon ja sillä ei ole väliä, kuinka se on toteutettu, käytän `string`-tyyppiä, joka on osa perus-C++-kirjastoa. Sinun kannattaisi myös tehdä niin. Haluan kuitenkin usein osoittaa kuinka C++-kieli käyttäytyy, ja näissä tapauksissa minun pitää näyttää hieman toteutuksen koodia. Käytän tällöin (epästandardi) `String`-luokkaa. Sinun pitäisi ohjelmoijana käyttää perus `string`-tyyppiä aina kun tarvitset merkkijono-olion; päivät, jolloin kehität oman merkkijonoluokkasi C++-kielen siirtymämuotona ovat takana päin. Sinun pitää silti ymmärtää asiat, jotka liittyvät luokkien, kuten `string`, kehitykseen. `String` on sopiva tuohon tarkoitukseen (ja vain tuohon tarkoitukseen). Mitä tulee raakoihin `char*`-pohjaisiin merkkijonoihin, sinun ei pitäisi käyttää noita antiikkisia jäänteitä, paitsi jos sinulla on erittäin hyvä syy käyttää niitä. Hyvin toteutetut `string`-tyypit voivat nyt olla

parempia kuin `char*`-pohjaiset merkkijonot lähes joka tavalla - mukaan lukien tehokkuus (katso Kohta 49).

Kaksi seuraava termiä, joiden kanssa painiskelemme, ovat *alustus* (initialization) ja *sijoitus* (assignment). Olion alustus tapahtuu silloin, kun sille annetaan arvo ensimmäistä kertaa. Luokan olioille tai muodostinfunktiolla varustetuille struktuureille alustus suoritetaan aina kutsumalla muodostinfunktioa. Tämä on erilaista kuin olion sijoitus, joka tapahtuu silloin, kun oliolle, joka on jo valmiiksi alustettu, annetaan uusi arvo:

```
string s1;                    // alustus
string s2("Hello");           // alustus
string s3 = s2;               // alustus

s1 = s3;                      // sijoitus
```

Puhtaasti toiminnallisesta näkökulmasta ero alustamisen ja sijoituksen välillä on se, että aiempi on muodostinfunktion suorittama, kun taas jälkimmäinen `operator=`-funktion suorittama. Nämä kaksi prosessia vastaavat toisin sanoen eri funktiokutsuihin.

Syy jakoon on siinä, että näiden kahden funktiotyypin täytyy murehtia erilaisista asioista. Muodostinfunktioiden täytyy yleensä tarkistaa argumenttiensa oikeellisuus, siinä missä useimmat sijoitusoperaattorit voivat olla varmoja siitä, että niiden argumenttinsa ovat laillisia (koska ne on jo muodostettu). Toisaalta muodostuksen alla olevaan sijoituksen kohteeseen voi jo olla varattuna resursseja, toisin kuin olioon. Nämä resurssit täytyy tyypillisesti vapauttaa ennen kuin uusia resursseja voidaan sijoittaa. Usein yksi näistä resursseista on muisti. Ennen kuin sijoitusoperaattori voi varata muistia uudelle arvolle, sen täytyy ensin vapauttaa muisti, joka oli varattu vanhalle arvolle.

Tässä on tapa, jolla String-muodostinfunktio ja sijoitusoperaattori voidaan toteuttaa:

```
// mahdollinen String-muodostin
String::String(const char *value)
{
  if (value) {                // jos arv. osoitin ei null
    data = new char[strlen(value) + 1];
    strcpy(data,value);
  }
  else {                      // hoid null-arv. osoitin†
    data = new char[1];
```

† Minun `String`-luokkani muodostinfunktio, joka ottaa `const char*` -tyyppisen argumentin, hoitaa asian, kun tyhjä osoitin välitetään sille, mutta perus-`string`-tyypin ei vaadita olevan niin suvaitseva. Yritykset luoda `string` tyhjästä osoittimesta tuottavat määrittelemättömiä tuloksia. On kuitenkin turvallista luoda `string`-olio tyhjästä `char*`-pohjaisesta merkkijonosta, toisin sanoen `""`.

```
    *data = '\0';                // lisää loppumerk. mjonoon
  }
}

// mahdollinen String sijoitusoperaattori
String& String::operator=(const String& rhs)
{
  if (this == &rhs)
    return *this;                // katso Kohta 17

  delete [] data;                // poista vanha muisti
  data =                         // varaa uutta muistia
    new char[strlen(rhs.data) + 1];
  strcpy(data, rhs.data);

  return *this;                  // katso Kohta 15
}
```

Huomaa kuinka muodostinfunktion täytyy tarkistaa parametrinsa oikeellisuus ja kuinka sen täytyy nähdä vaivaa varmistaakseen, että jäsen `data` on oikein alustettu, toisin sanoen se osoittaa `char*`-osoittimeen, joka on oikein päätetty null-merkillä. Sijoitusoperaattori edellyttää toisaalta, että sen parametri on laillinen. Se keskittyy patologisten ehtojen tunnistamiseen, kuten itseensä kohdistuvien sijoitusten tarkistamiseen (katso Kohta 17) ja vanhan muistin vapauttamiseen ennen kuin se varaa uutta muistia. Erot näiden kahden funktion välillä ovat esimerkkinä eroista olion alustamisen ja olion sijoittamisen välillä. Muuten, jos `"[]"` -esitysmuoto `delete`-operaattorin käytössä on uusi asia sinulle, Kohdan 5 pitäisi häivyttää mahdollinen hämmennys.

Viimeinen keskustelua vaativa termi on asiakas (client). Asiakas on ohjelmoija, joka käyttää kirjoittamaasi koodia. Kun puhun tässä kirjassa asiakkaista, viittaan ihmisiin, jotka tutkivat koodiasi, yrittäen hahmottaa mitä se tekee; ihmisiin, jotka lukevat luokkamäärityksesi, yrittäen määrittää, haluavatko he periä luokistasi; ihmisiin, jotka tutkivat työtapapäätöksiäsi, toivoen voimansa poimia oivalluksia perusteluihinsa.

Et ehkä ole tottunut ajattelemaan asiakkaitasi, mutta kulutan aika paljon aikaa vakuuttaakseni sinulle, että tekisit heidän elämänsä mahdollisimman helpoksi. Sinähän olet muiden ihmisten kehittämien ohjelmien asiakas. Etkö haluaisi, että nämä ihmiset tekisivät asiat helpoksi sinulle? Sitä paitsi, löydät jonain päivänä itsesi siinä epämukavassa asemassa, jossa joudut käyttämään omaa koodiasi, jolloin tulet olemaan oma asiakkaasi!

Käytän tässä kirjassa kahta rakennetta, jotka eivät ehkä ole tuttuja sinulle. Molemmat ovat suhteellisen tuoreita lisäyksiä C++-kieleen. Ensimmäinen on `bool`-tyyppi, jolla on arvoinaan avainsanat `true` ja `false`. Tämä on tyyppi, jonka nyt palauttavat sisäänrakennetut suhteelliset operaattorit (eli `<`, `>`, `==`) ja ne on testattu `if`-, `for`-, `while`- ja `do`-lauseiden ehto-osassa. Jos kääntäjäsi eivät vielä ole toteuttaneet

`bool`-tyyppiä, helppo tapa approksimoida se on käyttää typedef-komentoa `bool`-tyypille ja vakio-olioita `true`- ja `false`-avainsanoille:

```
typedef int bool;

const bool false = 0;
const bool true = 1;
```

Tämä on yhteensopiva C- ja C++-kielen perinteisen merkitysopin kanssa. Tätä approksimaatiota käyttävien ohjelmien käyttäytyminen ei muutu, kun ne siirretään `bool`-tyyppiä tukeviin kääntäjiin.

Toinen uusi rakenne on itse asiassa kolme rakennetta, muunnosmuodot `static_cast`, `const_cast`, `dynamic_cast` ja `reinterpret_cast`. Perinteinen C-tyylinen muunnos näyttää tältä:

```
(type) expression                // muunnos ilmaisu, joka
                                 // on tyyppiä type
```

Uudet muunnokset näyttävät tältä:

```
static_cast<type>(expression)  // muunnos ilmaisu, joka on
                               // tyyppiä type
const_cast<type>(expression)

dynamic_cast<type>(expression)

reinterpret_cast<type>(expression)
```

Nämä muunnoksen erilaiset muodot sopivat eri tarkoituksiin:

- `const_cast` on suunniteltu muuntamaan olioiden ja osoittimien `const`-tyyppisyys, aihe jota tutkin Kohdassa 21.
- `dynamic_cast` käytetään suorittamaan "turvallinen alasmuunnos", jota tutkimme Kohdassa 39.
- `reinterpret_cast` on suunniteltu muunnoksille, jotka saavat aikaan toteutuksesta riippuvaisia tuloksia, toisin sanoen muunnokset funktion osoittimen tyyppien välillä. (Et todennäköisesti tarvitse `reinterpret_cast`-muunnosta kovin usein. En käytä sitä ollenkaan tässä kirjassa.)
- `static_cast` on eräänlainen kaikkeensopiva muunnos. Sitä käytetään silloin, kun mikään muu muunnos ei ole sopiva. Se on lähimpänä perinteisen C-tyylisen muunnoksen tarkoitusta.

Perinteiset muunnokset ovat edelleen sallittuja, mutta uudet muunnosmuodot ovat suositeltavimpia. Sekä ihmisten että työkalujen, kuten `grep`, on helpompi tunnistaa ne koodissa ja jokaisen muunnosmuodon määritetty tarkoitus on mahdollistaa kääntäjien käännösvirheiden diagnostisoinnin. Esimerkiksi, vain `const_cast`-muunnosta voidaan käyttää muuntamaan `const`-tyyppisyys. Jos yrität muuntaa

olion tai osoittimen `const`-tyyppisyyden käyttämällä yhtä muista uusista muunnoksista, muunnosilmaisusi ei käänny.

Jos haluat lisätietoja uusista muunnoksista, kysy neuvoa C++-kielen viimeaikaisista johdantokirjoista tai katso kirjani *More Effective C++* Kohta 2. (Löydät yleisesityksen kirjasta *More Effective C++* tämän kirjan sivuilta 237-238.)

Tämän kirjan koodiesimerkeissä olen yrittänyt valita mielekkäitä nimiä olioille, luokille, funktioille ja niin edelleen. Monet kirjat käyttävät tunnisteissaan lyhytaikaisuutta, pitäen sitä nokkelana, mutta en ole niin kiinnostunut nokkelana kuin olen älykkäänä olemisesta. Olen siksi pyrkinyt rikkomaan perinteen, jossa ohjelmointikielten kirjoissa käytetään kryptisiä tunnisteita. Joka tapauksessa olen silloin tällöin langennut houkutukseen käyttää kahta parametrisuosikkiani, ja niiden tarkoitukset eivät ehkä heti ole ilmeisiä, varsinkaan jos et ole koskaan kuluttanut aikaa kääntäjille kirjoittavassa kahlejoukkueessa.

Nimet ovat `lhs` ja `rhs`, ja ne ovat lyhenteitä sanoista "left-hand side" ja "right-hand side". Käytän niitä parametrien niminä funktioille, joilla toteutetaan binäärioperaattoreita, varsinkin `operator==` ja aritmeettisia operaattoreita kuten `operator*`. Jos esimerkiksi `a` ja `b` ovat olioita, joilla esitetään suhdelukuja, ja jos suhdeluvut voidaan kertoa ei-jäsen -tyyppisellä `operator*`-funktiolla, ilmaisu

```
a * b
```

on vastaava kuin funktiokutsu

```
operator*(a, b)
```

Kuten huomaat Kohdasta 23, esittelen `operator*`-funktion tällä tavalla:

```
const Rational operator*(const Rational& lhs,
                         const Rational& rhs);
```

Kuten näet, vasemmanpuoleinen operandi `a` tunnetaan funktion sisällä nimellä `lhs`, ja oikeanpuoleinen operandi tunnetaan nimellä `rhs`.

Olen päättänyt lyhentää osoittimien nimet tämän säännön mukaisesti: osoitinta olioon, joka on tyyppiä `T`, kutsutaan usein `pt`, "osoitin `T`-olioon". Tässä on muutamia esimerkkejä:

```
string *ps;                    // ps = osoit. merkkijonoon

class Airplane;
Airplane *pa;                  // pa = os. Airplane-luok.

class BankAccount;
BankAccount *pba;              // pba = os. BankAccount-
                               // luokkaan
```

Käytän samankaltaista merkintätapaa viittauksiin. Tämä tarkoittaa sitä, että `rs` voisi olla viittaus `string`-muuttujaan ja `ra` voisi olla viittaus `Airplane`-luokkaan.

Käytän satunnaisesti nimeä `mf`, kun puhun jäsenfunktioista.

On mahdollista, että aiheutuu hämmennystä siitä, että kun mainitsen C-ohjelmointikielen tässä kirjassa, tarkoitan ISO/ANSI-versiota C-kielestä, enkä vanhempaa kevyemmin tyypitettyä, "klassista" C-kieltä.