Osa IV

Tehotyökalut

Oppitunnit

Osa
IV

13. oppitunti

Kehittyneet funktiot

Luvussa 5, "Funktiot", käsiteltiin funktioiden peruskäyttöä. Nyt, kun tiedät, kuinka osoittimet ja viittaukset toimivat, voimme jatkaa funktioiden käsittelyä. Tässä luvussa käsittelemme seuraavia aiheita:

- Kuinka jäsenfunktioita ylikuormitetaan
- Kuinka kirjoitetaan funktioita, jotka tukevat luokkia, joissa on dynaamisesti varattuja muuttujia

Ylikuormitetut jäsenfunktiot

Luvussa 5 opit toteuttamaan funktion monimuotoisuuden eli ylikuormituksen kirjoittamalla samannimisiä funktioita, jotka ottavat erilaisia parametreja. Luokan jäsenfunktioita voidaan myöskin ylikuormittaa.

Listauksen 13.1 Rectangle-luokalla on kaksi DrawShape()-funktiota. Toinen ei ota lainkaan parametreja, vaan piirtää suorakaiteen luokan nykyisillä arvoilla. Toinen funktio taas ottaa kaksi parametria, leveyden ja pituuden, ja piirtää suorakaiteen noilla arvoilla ohittaen luokan nykyiset arvot.

Listaus 13.1. Jäsenfunktioiden ylikuormittaminen.

```
1:  //Listaus 13.1 Metodien ylikuormitus.
2:  #include <iostream.h>
3:
4:  typedef unsigned short int USHORT;
5:  enum BOOL { FALSE, TRUE};
6:
7:  // Rectangle-luokan esittely
8:  class Rectangle
9:  {
10: public:
11:   // muodostimet
12:   Rectangle(USHORT width, USHORT height);
13:   ~Rectangle(){}
14:
15:   // ylikuormitettu DrawShape
16:   void DrawShape() const;
17:   void DrawShape(USHORT aWidth, USHORT aHeight) const;
18:
19: private:
20:   USHORT itsWidth;
21:   USHORT itsHeight;
22: };
23:
24: //Muodostimen määrittely
25: Rectangle::Rectangle(USHORT width, USHORT height)
26: {
27:   itsWidth = width;
28:   itsHeight = height;
29: }
30:
31:
32: // Ylikuormitettu DrawShape - ei ota arvoja
33: // Piirtää perustuen nykyisiin arvoihin
34: void Rectangle::DrawShape() const
35: {
36:   DrawShape( itsWidth, itsHeight);
37: }
38:
39:
40: // Ylikuormitettu DrawShape - ottaa 2 arvoa
41: // Piirtää parametrien mukaan.
42: void Rectangle::DrawShape(USHORT width, USHORT height) const
43: {
44:   for (USHORT i = 0; i<height; i++)
45:   {
46:     for (USHORT j = 0; j< width; j++)
47:     {
48:       cout << "*";
49:     }
50:     cout << "\n";
51:   }
52: }
53:
54: // Pääohjelma
```

```
55: int main()
56: {
57:   // alustusarvot 30,5
58:
59:   Rectangle theRect(30,5);
60:   cout << "DrawShape(): \n";
61:   theRect.DrawShape();
62:   cout << "\nDrawShape(40,2): \n";
63:   theRect.DrawShape(40,2);
64:   return 0;
65: }
```

Tulostus

```
DrawShape():
**************************************
**************************************
**************************************
**************************************
**************************************

DrawShape(40,2):
************************************************
************************************************
```

Huom! Listauksessa viedään leveys ja korkeus usealle funktiolle. Huomaa, että joskus leveys viedään ensin ja joskus taas korkeus.

Analyysi

Tärkeä koodilohko on riveillä 16-17, missä DrawShape() ylikuormitetaan. Noiden metodien toteutus on riveillä 32-52. Funktio, joka ei ota parametreja, kutsuu versiota, joka ottaa kaksi parametria, jotka ovat nykyiset jäsenmuuttujat. Yritä kaikin tavoin välttää koodin toistoa kahdessa funktiossa. Muutoin muutosten tekeminen jompaan kumpaan funktioon voi olla vaikeaa ja virhealtista.

Rivien 55-63 pääohjelma luo Rectangle-olion ja kutsuu sitten DrawShape()-funktiota: ensimmäisessä kutsussa ei ole parametreja, kun taas toisessa kutsussa käytetään kahta parametria, jotka ovat tyypiltään unsigned short -kokonaislukuja.

Kääntäjä päättelee, kumpaa metodia se kutsuu parametrien määrästä ja tyypeistä. Voisit kuvitella vielä kolmannen ylikuormitetun metodin nimeltä DrawShape(), joka ottaa yhden mitta-arvon sekä järjestetyn luettelon, joka kertoo, onko käyttäjä valinnut leveyden vai korkeuden.

Oletusarvojen käyttäminen

Tavallisilla funktioilla voi olla yksi tai useampia oletusarvoja ja samoin myös luokan metodeilla. Niiden kohdalla sovelletaan samoja sääntöjä oletusarvojen esittelyyn, kuten listaus 13.2 osoittaa.

Listaus 13.2. Oletusarvojen käyttäminen.

```
1:  //Listaus 13.2 Oletusarvot
2:  #include <iostream.h>
3:
4:  typedef unsigned short int USHORT;
5:  enum BOOL { FALSE, TRUE};
6:
7:  // Rectangle-luokan esittely
8:  class Rectangle
9:  {
10: public:
11:   // muodostimet
12:   Rectangle(USHORT width, USHORT height);
13:   ~Rectangle(){}
14:   void DrawShape(USHORT aWidth, USHORT aHeight, BOOL UseCurrentVals  FALSE) const;
15: private:
16:   USHORT itsWidth;
17:   USHORT itsHeight;
18: };
19:
20: //Muodostimen toteutus
21: Rectangle::Rectangle(USHORT width, USHORT height):
22: itsWidth(width),       // alustukset
23: itsHeight(height)
24: {}                     // tyhjä runko
25:
26:
27: // oletusarvot 3. parametrilla
28: void Rectangle::DrawShape(
29: USHORT width,
30: USHORT height,
31: BOOL UseCurrentValue
32: ) const
33: {
34:   int printWidth;
35:   int printHeight;
36:
37:   if (UseCurrentValue == TRUE)
38:   {
39:     printWidth = itsWidth;       // käytä nyk. arvoja
40:     printHeight = itsHeight;
41:   }
42:   else
43:   {
44:     printWidth = width;         // käytä param. arvoja
45:     printHeight = height;
46:   }
47:
48:
49:   for (int i = 0; i<printHeight; i++)
50:   {
51:     for (int j = 0; j< printWidth; j++)
52:     {
```

```
53:       cout << "*";
54:     }
55:     cout << "\n";
56:   }
57: }
58:
59: // Pääohjelma
60: int main()
61: {
62:   // alustusarvot 30,5
63:   Rectangle theRect(30,5);
64:   cout << "DrawShape(0,0,TRUE)...\n";
65:   theRect.DrawShape(0,0,TRUE);
66:   cout <<"DrawShape(40,2)...\n";
67:   theRect.DrawShape(40,2);
68:   return 0;
69: }
```

Tulostus

```
DrawShape(0,0,TRUE)…
********************************
********************************
********************************
********************************
********************************

DrawShape(40,2)…
************************************************
************************************************
```

Analyysi

Listauksessa 13.2 korvataan ylikuormitettu DrawShape() yksittäisellä funktiolla, jolla on oletusparametrit. Funktio esitellään riveillä 14-15 ottamaan kolme parametria. Kaksi ensimmäistä parametria, aWidth ja aHeight, ovat USHORT-tyyppiä ja kolmas, UseCurrentVals on boolean-tyyppinen (oletusarvo FALSE).

Huom! Boolean-muuttujien arvo on joko tosi tai epätosi. C++ -kielessä on arvo 0 epätosi ja kaikki muut arvot tosia.

Funktion toteutus alkaa riviltä 28. Kolmas parametri, UseCurrentVals, testataan. Jos arvo on TRUE, käytetään jäsenmuuttujia itsWidth ja itsHeight asettamaan paikalliset muuttujat printWidth ja printHeight.

Jos UseCurrentValue on FALSE, joko oletuksena tai käyttäjän asettamana, käytetään kahta ensimmäistä parametria asettamaan printWidth ja printHeight.

Huomaa, että jos UseCurrentVals on TRUE, ei kahta muuta parametrin arvoa käytetä.

Valitseminen oletusarvojen ja ylikuormitettujen funktioiden välillä

Listaukset 13.1 ja 13.2 toteuttavat samat tehtävät, mutta listauksen 13.1 ylikuormitettuja funktioita on helpompi ymmärtää ja luonnollisempi käyttää. Jos vielä tarvitaan kolmatta muunnosta, jossa käyttäjä ehkä haluaa antaa joko leveyden tai korkeuden, mutta ei molempia, on helppo laajentaa ylikuormitettuja funktioita. Oletusarvoista tulee sen sijaan helposti liian monimutkaisia kehitettäessä uusia variaatioita.

Kuinka sitten voisit päättää, käytätkö oletusarvoja vai ylikuormitettuja funktioita? Seuraavassa on muutama peukalosääntö:

Harkitse ylikuormittamista silloin,

- ❏ kun ei ole olemassa mitään järkevää oletusarvoa.
- ❏ kun tarvitset erilaisia algoritmeja.
- ❏ kun sinun on tuettava vaihtelevia tyyppejä parametriluettelossa.

Oletusmuodostin

Kuten luvussa 6, "Perusluokat", kerrottiin, muodostetaan oletusmuodostin silloin, kun ohjelmassa ei esitellä ulkoisesti muodostinta. Oletusmuodostin ei ota parametreja, eikä se tee mitään. Ohjelmassa voidaan kuitenkin laatia oma oletusmuodostin, joka ei ota argumentteja, mutta voi asettaa olion halutulla tavalla.

C++ -kielen tarjoama muodostin on oletuksellinen, mutta sääntöjen mukaan on jokainen muodostin, joka ei ota parametreja, oletusmuodostin. Tämä voi kuulostaa hieman hämäännyttävältä, mutta asiayhteys kertoo aina selvästi, kummasta on kysymys.

Muista, että kehittäessäsi minkä tahansa muodostimen, ei kääntäjä tee oletusmuodostinta. Jos siis haluat muodostimen, joka ei ota parametreja ja olet kehittänyt muita muodostimia, on sinun tehtävä myös oletusmuodostin itse.

Muodostimien ylikuormitus

Muodostinta käytetään olion luomiseen. Esimerkiksi Rectangle-muodostin luo suorakaideolion (eli suorakaiteen). Ennen muodostimen ajamista, ei suorakaidetta ole. Muodostimen ajamisen jälkeen on käytössä valmis Rectangle-olio.

Muodostimia, kuten kaikkia jäsenfunktioita, voidaan ylikuormittaa. Ylikuormituksen mahdollisuus lisää tehoa ja joustavuutta.

Saatat haluta esimerkiksi suorakaideolion, jolla on kaksi muodostinta: ensimmäinen ottaa parametreikseen pituuden ja leveyden ja muodostaa vastaavan suorakaiteen. Toinen ei ota lainkaan parametreja vaan muodostaa oletussuorakaiteen. Kääntäjä valitsee oikean muodostimen samoin periaattein kuin ylikuormitetun funktion: päätös perustuu parametrien määrään ja tyyppiin.

Vaikka muodostimia voidaankin ylikuormittaa, niin ei voida tehdä tuhoajafunktioille. Niillä on aina sama nimi: luokan nimi, tilde-merkki, eikä yhtään parametria.

Olioiden alustaminen

Toistaiseksi olet sijoittanut arvot olioiden jäsenmuuttujiin muodostimen rungossa. Muodostimet luodaan kuitenkin kahdessa vaiheessa: alustusvaihetta seuraa muodostimen runko.

Useimmat muuttujat voidaan asettaa jommassakummassa vaiheessa: ne voidaan alustaa alustusvaiheessa tai niihin voidaan sijoittaa arvo rungossa. On selvempää ja usein tehokkaampaa alustaa jäsenmuuttujat alustusvaiheessa. Seuraava esimerkki selvittää, kuinka jäsenmuuttujat alustetaan:

```
CAT():  // muodostimen nimi ja parametrit
itsAge(5),  // alustusluettelo
itsWeight(8)
{ } // muodostimen runko
```

Muodostimen parametriluettelon jälkeen laitetaan kaksoispiste. Sen jälkeen tulee jäsenmuuttuja sekä sulkumerkit. Sulkumerkkien sisälle laitetaan alustusarvo. Jos alustuksia on useita, ne erotetaan toisistaan pilkuilla.

Muista, että viittaukset ja vakiot on alustettava eikä niihin voida sijoittaa arvoja. Jos jäsentietoina on viittauksia tai vakioita, ne tulee alustaa edellä kuvatulla tavalla.

Aiemmin kerrottiin, että alustaminen on tehokkaampaa kuin sijoittaminen. Ymmärtääksesi tämän on sinun tunnettava kopiomuodostin.

Kopiomuodostin

Oletusmuodostimen ja tuhoajafunktion lisäksi kääntäjällä on käytössään myös oletuskopiomuodostin. Kopiomuodostinta kutsutaan joka kerta, kun oliokopio tehdään. Kun olio viedään arvona, joko funktioon tai funktion palautuksena, luodaan tilapäinen kopio oliosta. Jos olio on käyttäjän määrittelemä, kutsutaan luokan kopiomuodostinta. Kaikki kopiomuodostimet ottavat yhden parametrin: viittauksen saman luokan olioon. On hyvä laatia vakioviittaus, koska muodostimen ei tule muuttaa sille vietyä oliota. Esimerkiksi:

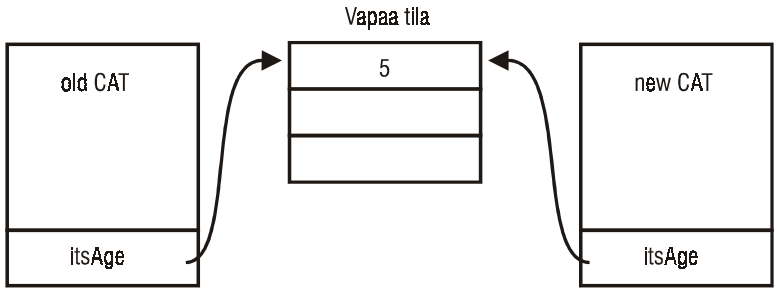
```
CAT(const CAT &theCat);
```

Edellä CAT-muodostin ottaa parametrikseen vakioviittauksen olemassaolevaan CAT-olioon. Kopiomuodostimen tavoitteena on tehdä theCat-kopio.

Oletuskopiomuodostin yksinkertaisesti kopioi sille viedyn olion jäsenmuuttujat uuden olion jäsenmuuttujiksi. Kyseessä on jäsenkohtainen kopio ja vaikka tämä menettely toimiikin useimpien jäsenmuuttujien kohdalla, se kaatuu helposti sellaisten jäsenmuuttujien kohdalla, jotka ovat osoittimia vapaaseen muistitilaan.

Uusi käsite Jäsenkohtainen kopiointimenettely kopioi olion jäsenmuuttujien tiedot toiseen olioon. Molempien olioiden osoittimet eivät enää osoita samaan muistialueeseen. Tarkka (syvä) kopio sen sijaan kopioi keossa olevat arvot uudelleen varattuun muistiin.

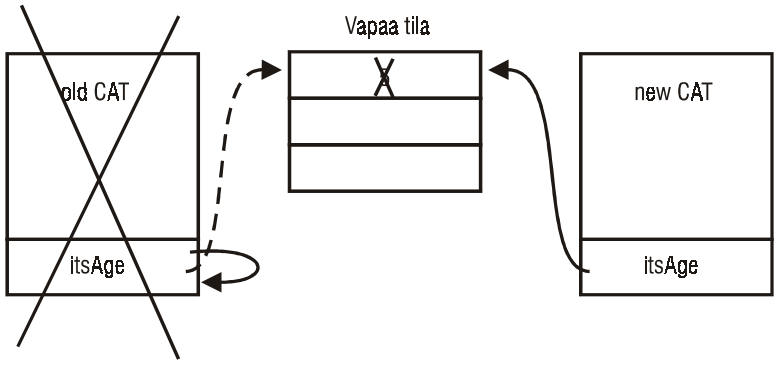
Jos CAT-luokassa on jäsenmuuttuja itsAge, joka osoittaa muistissa olevaan kokonaislukuun, oletuskopiomuodostin kopioi olion itsAge-jäsenmuuttujan uuden olion itsAge-jäsenmuuttujaksi. Nuo kaksi oliota osoittavat sitten samaan muistiin, kuten kuva 13.1 näyttää.



Kuva 13.1. Oletuskopiomuodostimen käyttö.

Tällainen menettely johtaa hankaluuksiin silloin, kun jompikumpi CAT-olio ei ole käytettävyysalueella. Kun olion käytettävyysalue päättyy, kutsutaan tuhoajafunktiota, joka yrittää vapauttaa varatun muistin.

Olettakaamme, että alkuperäinen olio ei enää ole näkyvyysalueella. Sen tuhoajafunktio vapauttaa varatun muistin. Kopio osoittaa edelleenkin tuohon muistiin ja sen yrittäessä käsitellä tuota muistia ohjelma kaatuu - jos sinulla on onnea. Kuva 13.2 havainnollistaa tätä ongelmaa.



Kuva 13.2. Villin osoittimen luonti.

Ratkaisuna tähän ongelmaan on oman kopiomuodostimen luominen ja muistin varaaminen kopiolle. Kun muisti on varattu, voidaan vanhat arvot kopioida uuteen muistiin. Tätä kutsutaan syväksi kopioksi. Listaus 13.3 havainnollistaa tätä menettelyä.

Listaus 13.3. Kopiomuodostimet.

```
1:  // Listaus 13.3
2:  // Kopiomuodostimet
3:
4:  #include <iostream.h>
5:
6:  class CAT
7:  {
8:  public:
9:    CAT();                          // oletusmuodostin
10:   CAT (const CAT &);     // kopiomuodostin
11:   ~CAT();                          // tuhoaja
12:   int GetAge() const { return *itsAge; }
13:   int GetWeight() const { return *itsWeight; }
14:   void SetAge(int age) { *itsAge = age; }
15:
16: private:
17:   int *itsAge;
18:   int *itsWeight;
19: };
20:
21: CAT::CAT()
22: {
23:   itsAge = new int;
24:   itsWeight = new int;
25:   *itsAge = 5;
```

```
26:   *itsWeight = 9;
27: }
28:
29: CAT::CAT(const CAT & rhs)
30: {
31:   itsAge = new int;
32:   itsWeight = new int;
33:   *itsAge = rhs.GetAge();
34:   *itsWeight = rhs.GetWeight();
35: }
36:
37: CAT::~CAT()
38: {
39:   delete itsAge;
40:   itsAge = 0;
41:   delete itsWeight;
42:   itsWeight = 0;
43: }
44:
45: int main()
46: {
47:   CAT frisky;
48:   cout << "frisky's age: " << frisky.GetAge() << endl;
49:   cout << "Setting frisky to 6...\n";
50:   frisky.SetAge(6);
51:   cout << "Creating boots from frisky\n";
52:   CAT boots(frisky);
53:   cout << "frisky's age: " <<     frisky.GetAge() << endl;
54:   cout << "boots' age: " << boots.GetAge() << endl;
55:   cout << "setting frisky to 7...\n";
56:   frisky.SetAge(7);
57:   cout << "frisky's age: " <<     frisky.GetAge() << endl;
58:   cout << "boot's age: " << boots.GetAge() << endl;
59:   return 0;
60: }
```

Tulostus

```
frisky's age: 5
Setting frisky to 6…
Creating boots from frisky
frisky's age: 6
setting frisky to 7…
frisky's age: 7
boots' age: 6
```

Analyysi

Rivit 6-19 esittelevät CAT-luokan. Huomaa, että rivillä 9 on oletusmuodostin ja rivillä 10 kopiomuodostin.

Rivillä 17 ja 18 esitellään kaksi jäsenmuuttujaa, jotka ovat kokonaislukuosoittimia. Yleensä luokalla ei ole tarvetta tallentaa int-jäsenmuuttujia osoittimina, mutta tämä tehtiin nyt havainnollistamaan jäsenmuuttujien hallintaa vapaassa muistissa.

Oletusmuodostin riveillä 21-27 varaa tilaa vapaasta muistista noille kahdelle int-muuttujalle ja sijoittaa sitten arvot niihin.

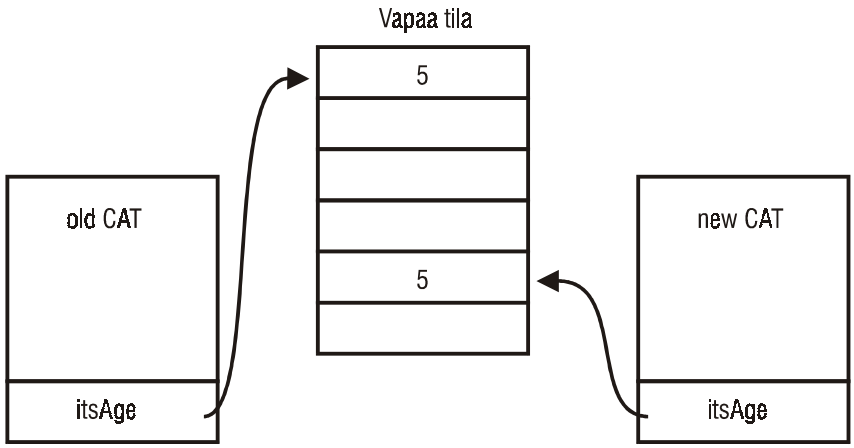
Kopiomuodostin alkaa riviltä 29. Huomaa, että parametri on rhs. On yleistä viitata kopiomuodostimen parametriin tunnuksella rhs, joka tarkoittaa 'right hand side'. Rivien 33 ja 34 sijoituksista näet, että parametrina vietävä kohde on yhtäsuuruus-merkin oikealla puolella. Se toimii näin:

- Riveillä 31-32 varataan tilaa muistista. Riveillä 33-34 muistipaikkaan sijoitetaan olemassaolevan CAT-olion arvot.
- Parametri rhs on CAT-olio, joka viedään kopiomuodostimelle vakioviittauksena. Jäsenfunktio rhs.GetAge() palauttaa itsAge-osoittimen osoittamasta muistista arvon. Kuten CAT-oliollakin rhs-oliolla on kaikki CAT-jäsenmuuttujat.
- Kun kopiomuodostinta kutsutaan luomaan uusi CAT-olio, olemassa oleva CAT viedään parametrina. Uusi CAT-olio voi viitata omiin jäsenmuuttujiinsa suoraan; kuitenkin sen on käsiteltävä rhs:n jäsenmuuttujia julkisilla käsittelyfunktioilla.

Kuva 13.3 kertoo kaaviona, mitä tapahtuu. Olemassa olevan CAT-olion osoittamat arvot kopioidaan uudelle CAT-oliolle varattuun muistiin.

Rivillä 47 luodaan CAT-olio, Frisky. Sen ikä tulostetaan ja iäksi sijoitetaan uusi arvo, 6, rivillä 50. Rivillä 52 luodaan uusi CAT-olio, boots, kopiomuodostimen avulla, jolle viedään Frisky. Jos Frisky olisi viety parametrina funktiolle, olisi kääntäjä kutsunut tuota kopiomuodostinta.

Riveillä 53-54 tulostetaan kummankin CAT-olion iät. Nähdään, että bootsin ikä on 6, kuten Friskynkin, eikä siis oletusarvo 5. Rivillä 56 sijoitetaan Friskyn iäksi 7 ja iät tulostetaan uudelleen. Tällä kertaa Friskyn ikä on 7, mutta bootsin edelleen 6, mikä havainnollistaa sitä, että oliot tallennetaan erillisille muistialueille.



Kuva 13.3. Havaintoesitys syvästä kopiosta.

Kun CAT-olioiden elinikä päättyy, niiden tuhoajafunktioita kutsutaan automaattisesti. CAT-tuhoajafunktion toteutus on riveillä 37-43. delete-operaattoria käytetään kummallekin osoittimelle, itsAge ja itsWeight, jolloin varattu muisti vapautetaan muuhun käyttöön. Turvallisuuden vuoksi osoittimiin sijoitetaan vielä NULL-arvot.

Yhteenveto

Tässä luvussa käsiteltiin luokkien jäsenfunktioiden ylikuormittamista. Opit myös viemään oletusarvot funktioille ja päättelemään, milloin käyttää oletusarvoja ja milloin taas ylikuormittamista.

Muodostimien ylikuormittaminen mahdollistaa joustavien luokkien luomisen, joita voidaan luoda muista olioista. Olioiden alustaminen tapahtuu muodostimen alustusvaiheessa, mikä on tehokkaampaa kuin arvojen sijoittaminen muodostimen rungossa.

Kääntäjä tarjoaa kopiomuodostimen, jos omaa ei luoda, mutta se tekee vain jäsenkohtaisen kopion luokasta. Jos luokan jäseninä on osoittimia vapaaseen tilaan, on tuo menettely korvattava niin, että oletuskohteelle varataan muistitilaa.

Kysymyksiä ja Vastauksia

K
Miksi oletusarvoja tulisi koskaan käyttää funktiota ylikuormitettaessa?
V
On helpompi ylläpitää yhtä kuin kahta funktiota. Usein on myös helpompi ymmärtää oletusparametrein varustettua funktiota kuin tutkia kahden funktion runkoja. Edelleen yhden funktion päivittäminen ja toisen päivityksen unohtaminen on yleinen virhe.

K
Kun tiedetään funktioiden ylikuormittamisen ongelmat, miksei aina käytetä oletusarvoja sen sijaan?
V
Ylikuormitetut funktiot tuovat mahdollisuuksia, joita oletusmuuttujilla ei saada aikaan, kuten tyyppien mukaan, ei pelkästään määrän mukaan vaihtelevat parametriluettelot.

K
Kuinka kopiomuodostinta luotaessa päätetään, mitä laitetaan alustukseen ja mitä taas muodostimen runkoon?
V
Yleissääntö on se, että alustuksessa tehdään niin paljon kuin mahdollista eli jäsenmuuttujat alustetaan siinä vaiheessa. Joitakin toimintoja, kuten laskennat ja tulostukset, sijoitetaan runkoon.

K
Voiko ylikuormitetulla funktiolla olla oletusparametreja?
V
Kyllä. Nuo tehokkaat ominaisuudet voidaan yhdistää. Yhdellä tai useammalla ylikuormitetulla funktiolla voi olla omat oletusarvonsa, jotka asetetaan normaalien sääntöjen mukaan.