Osa
III

10. oppitunti

Kehittyneet osoittimet

Eräs tehokkaimpia C++ -työkaluja on mahdollisuus käsitellä tietokoneen muistia suoraan osoittimien avulla. Tässä luvussa käsitelläänkin seuraavia aiheita:

- Kuinka osoittimia käytetään tehokkaasti
- Kuinka estetään muistiongelmat osoittimia käytettäessä
- Kuinka vapaalle muistialueelle luodaan kohteita

Olioiden luominen vapaalle muistialueelle

Samalla lailla kuin int-osoittimen voit luoda minkä tahansa muunkin tyyppisen osoittimen. Jos olet esitellyt tyyppiä Cat olevan olion, voit luoda tuohon luokkaan osoittavan osoittimen ja luoda Cat-olion vapaalle muistialueelle samalla lailla kuin pinoonkin. Syntaksi on samanlainen kuin int-osoittimien kohdalla:

```
Cat *pCat = new Cat;
```

Lause kutsuu oletusmuodostinta - muodostinta, joka ei ota parametreja. Muodostinta kutsutaan aina, kun olio luodaan - joko pinoon tai kekoon.

Kohteiden tuhoaminen

Kun delete-operaattoria käytetään osoittimelle, joka osoittaa vapaalla muistialueella olevaan olioon, kutsutaan tuon olion tuhoajafunktiota ennen muistin vapauttamista. Listaus 10.1 havainnollistaa olioiden luomista ja tuhoamista vapaalla muistialueella.

Listaus 10.1. Olioiden luominen ja tuhoaminen vapaalla muistialueella.

```
1:  // Listaus 10.1
2:  // Kohteet vapaaseen muistiin
3:
4:  #include <iostream.h>
5:
6:  class SimpleCat
7:  {
8:  public:
9:    SimpleCat();
10:   ~SimpleCat();
11: private:
12:   int itsAge;
13: };
14:
15: SimpleCat::SimpleCat()
16: {
17:   cout << "Constructor called.\n";
18:   itsAge = 1;
19: }
20:
21: SimpleCat::~SimpleCat()
22: {
23:   cout << "Destructor called.\n";
24: }
25:
26: int main()
27: {
28:   cout << "SimpleCat Frisky...\n";
29:   SimpleCat Frisky;
30:   cout << "SimpleCat *pRags = new SimpleCat...\n";
31:   SimpleCat * pRags = new SimpleCat;
32:   cout << "delete pRags...\n";
33:   delete pRags;
34:   cout << "Exiting, watch Frisky go...\n";
35:   return 0;
36: }
```

Tulostus

```
SimpleCat Frisky
Constructor called.
SimpleCat *pRags = new SimpleCat…
Constructor called.
delete pRags…
Destructor called.
Exiting, watch Frisky go…
Destructor called.
```

Analyysi

Rivit 6-13 esittelevät luokan SimpleCat. Rivi 9 esittelee luokan SimpleCat muodostimen ja riveillä 15-19 on sen toteutus. Rivi 10 esittelee luokan SimpleCat tuhoajafunktion, jonka toteutus on riveillä 21-24.

Rivillä 29 luodaan Frisky pinoon, jolloin tarvitaan muodostinta. Rivillä 31 luodaan pRags-osoittimen osoittama SimpleCat kekoon, jolloin myöskin käytetään muodostinta. Rivillä 33 käytetään delete-operaattoria pRags-osoittimelle, jolloin käytetään tuhoajafunktiota. Kun funktio päättyy, loppuu Friskyn elinikä ja kutsutaan tuhoajafunktiota.

Luokan jäsenten käsittely

Paikallisesti luotujen Cat-olioiden tietojäseniä ja funktioita käsiteltiin pisteoperaattorin (.) avulla. Vapaalla muistialueella olevaa Cat-oliota käsitellään osoittimen uudelleenviittauksen kautta ja käyttämällä pisteoperaattoria. Esimerkiksi GetAge-funktiota voidaan käyttää seuraavasti:

```
(*pRags).GetAge();
```

Sulkumerkeillä varmistetaan, että pRags-osoittimella viitataan uudelleen ennen GetAge()-funktion käyttöä.

Koska tuo koodi näyttää hieman sekavalta, C++ tarjoaa suoremman operaattorin epäsuoraan käsittelyyn: osoitinoperaattorin (->), joka on siis väliviivan ja suurempi-merkin yhdistelmä. C++ käsittelee merkkejä yhtenä symbolina. Listaus 10.2 esittelee vapaalla muistialueella olevien olioiden jäsenten käsittelyä.

Listaus 10.2. Vapaalla muistialueella olevien olioiden jäsentiedon käsittely.

```
1:  // Listaus 10.2
2:  // Keossa olevien olioiden käsittely
3:
4:  #include <iostream.h>
5:
6:  class SimpleCat
7:  {
8:  public:
9:    SimpleCat() {itsAge = 2; }
```

```
10:    ~SimpleCat() {}
11:    int GetAge() const { return itsAge; }
12:    void SetAge(int age) { itsAge = age; }
13:  private:
14:    int itsAge;
15:  };
16:
17:  int main()
18:  {
19:    SimpleCat * Frisky = new SimpleCat;
20:    cout << "Frisky is " << Frisky->GetAge() << " years old\n";
21:    Frisky->SetAge(5);
22:    cout << "Frisky is " << Frisky->GetAge() << " years old\n";
23:    delete Frisky;
24:    return 0;
25:  }
```

Tulostus

```
Frisky is 2 years old
Frisky is 5 years old
```

Analyysi

Rivillä 19 luodaan SimpleCat vapaalle muistialueelle. Oletusmuodostin asettaa sen iäksi 2. GetAge()-metodia kutsutaan rivillä 20. Koska kyseessä on osoitin, käytetään osoitinoperaattoria (->) käsittelemään jäsentietoa ja funktioita. Rivillä 21 kutsutaan SetAge()-metodia ja rivillä 22 käytetään GetAge()-metodia uudelleen.

Jäsentieto vapaalla muistialueella

Yksi tai useampi luokan tietojäsenistä voi olla osoitin vapaalla muistialueella olevaan kohteeseen. Muistia voidaan varata muodostimessa tai metodeilla ja se voidaan vapauttaa tuhoajafunktiolla, kuten listaus 10.3 osoittaa.

Listaus 10.3. Osoittimet jäsentietoina.

```
1:   // Listaus 10.3
2:   // Osoittimet jäsenmuuttujiin
3:
4:   #include <iostream.h>
5:
6:   class SimpleCat
7:   {
8:   public:
9:     SimpleCat();
10:    ~SimpleCat();
11:    int GetAge() const { return *itsAge; }
12:    void SetAge(int age) { *itsAge = age; }
13:
14:    int GetWeight() const { return *itsWeight; }
15:    void setWeight (int weight) { *itsWeight = weight; }
16:
17:  private:
18:    int * itsAge;
19:    int * itsWeight;
20:  };
```

```
21:
22: SimpleCat::SimpleCat()
23: {
24:   itsAge = new int(2);
25:   itsWeight = new int(5);
26: }
27:
28: SimpleCat::~SimpleCat()
29: {
30:   delete itsAge;
31:   delete itsWeight;
32: }
33:
34: int main()
35: {
36:   SimpleCat *Frisky = new SimpleCat;
37:   cout << "Frisky is " << Frisky->GetAge() << " years old\n";
38:   Frisky->SetAge(5);
39:   cout << "Frisky is " << Frisky->GetAge() << " years old\n";
40:   delete Frisky;
41:   return 0;
42: }
```

Tulostus

```
Frisky is 2 years old
Frisky is 5 years old
```

Analyysi

Luokassa SimpleCat on kaksi jäsenmuuttujaa, molemmat ovat int-osoittimia (rivit 18-19). Muodostin (rivit 22-26) alustaa osoittimet vapaaseen muistiin ja oletusarvoihin.

Tuhoajafunktio (rivit 28-32) vapauttaa varatun muistin. Koska kyseessä on tuhoajafunktio, ei ole tarvetta asettaa näitä osoittimia null-osoittimiksi, koska ne eivät ole enää käytettävissä. Tässä onkin yksi selkeä paikka, jossa osoitinta ei aseteta null-osoittimeksi muistin vapauttamisen jälkeen.

Kutsuva funktio - nyt main() - ei tiedä, että itsAge ja itsWeight ovat osoittimia vapaaseen muistiin. Muistin hallinnan yksityiskohdat on kätketty luokan toteutukseen, kuten pitääkin tehdä.

Kun Frisky tuhotaan rivillä 40, kutsutaan tuhoajafunktiota. Tuhoajafunktio tuhoaa jokaisen jäsenosoittimensa. Jos nuo osoittimet vuorostaan osoittavat muihin käyttäjän määrittelemien luokkien olioihin, niiden muodostimia kutsutaan myöskin.

this-osoitin

Jokaisella luokan jäsenfunktiolla on kätketty parametri: this-osoitin. Se osoittaa yksittäiseen olioon. Siksi jokaisessa GetAge()- ja SetAge()-kutsussa on mukana this-osoitin kätkettynä parametrina.

this-osoittimen tehtävänä on osoittaa yksittäistä oliota, jonka metodia käytetään. Yleensä sitä ei tarvita; voit vain kutsua metodia ja asettaa jäsenmuuttujia. Joskus on kuitenkin voitava käsitellä itse oliota (esimerkiksi nykyisen olion osoittimen käyttöä varten). Juuri tällaisessa tilanteessa on this-osoitin tarpeellinen.

Tavallisesti ei this-osoitinta tarvita olion jäsenmuuttujien käsittelyyn tuon olion metodeista käsin. Voit kuitenkin kutsua this-osoitinta ulkoisesti, jos haluat. Näin tehdään listauksessa 10.4, joka todentaa this-osoittimen olemassaolon ja toiminnan.

Listaus 10.4. this-osoittimen käyttö.

```
1:  // Listaus 10.4
2:  // this-osoitin
3:
4:  #include <iostream.h>
5:
6:  class Rectangle
7:  {
8:  public:
9:    Rectangle();
10:   ~Rectangle();
11:   void SetLength(int length) { this->itsLength = length; }
12:   int GetLength() const { return this->itsLength; }
13:
14:   void SetWidth(int width) { itsWidth = width; }
15:   int GetWidth() const { return itsWidth; }
16:
17: private:
18:   int itsLength;
19:   int itsWidth;
20: };
21:
22: Rectangle::Rectangle()
23: {
24:   itsWidth = 5;
25:   itsLength = 10;
26: }
27: Rectangle::~Rectangle()
28: {}
29:
30: int main()
31: {
32:   Rectangle theRect;
33:   cout << "theRect is " << theRect.GetLength() << " feet long.\n";
34:   cout << "theRect is " << theRect.GetWidth() << " feet wide.\n";
35:   theRect.SetLength(20);
36:   theRect.SetWidth(10);
37:   cout << "theRect is " << theRect.GetLength()<< " feet long.\n";
38:   cout << "theRect is " << theRect.GetWidth()<< " feet wide.\n";
39:   return 0;
40: }
```

Tulostus

```
theRect is 10 feet long
theRect is 5 feet long
theRect is 20 feet long
theRect is 10 feet long
```

Analyysi

SetLength()- ja GetLength()-funktiot käyttävät this-osoitinta ulkoisesti käsitelläkseen Rectangle-olion jäsenmuuttujia. SetWidth()- ja GetWidth()-jäsenfunktiot taas eivät käytä this-osoitinta. Funktioiden käyttäytymisessä ei ole eroa, vaikkakin viimeksi mainittujen käyttö on selkeämpää.

Mihin this-osoitinta käytetään?

Jos edellä olisi kerrottu kaikki this-osoittimesta, ei sitä olisi juuri kannattanut ottaa esillekään. this-osoitin on kuitenkin osoitin eli siihen tallennetaan olion osoite. Juuri siksi se voi ollakin tehokas työkalu.

Myöhemmin tässä kirjassa, operaattoreiden ylikuormittamista käsiteltäessä, käytetään this-osoitinta käytännön tilanteissa. Toistaiseksi on tärkeää oppia tuntemaan tämä this-osoitin: se on osoitin, joka osoittaa olioon itseensä.

this-osoitinta ei tarvitse itse luoda tai tuhota; kääntäjä huolehtii noista toimenpiteistä.

Osoitinongelmia

Vaikeasti havaittavia ohjelmavirheitä voi syntyä, kun osoittimen osoittama muistialue vapautetaan delete-operaattorilla ja myöhemmin osoitinta yritetään käyttää uudelleen sijoittamatta siihen uutta kohdetta. Tilanne muistuttaa sitä, kun yrität ottaa puhelimella yhteyttä organisaatioon, mutta koko organisaatio on muuttanut pois. Välttämättä ei tapahdu mitään kauheampaa, puhelin vain soi jossakin oudossa paikassa. Puhelinnumero on saatettu kuitenkin antaa jo jollekin toiselle organisaatiolle.

Varmista, ettet käytä osoitinta sen jälkeen, kun olet kohdistanut siihen delete-operaation. Osoitin osoittaa edelleenkin vanhalle muistialueelle, mutta kääntäjä on saanut oikeuden sijoittaa kyseiselle muistialueelle uutta tietoa; osoittimen käyttäminen saattaa kaataa ohjelmasi. Ja vielä pahempaa, ohjelmasi saattaa toimia aivan moitteettomasti, mutta kaatuu kuitenkin myöhemmässä vaiheessa. Ohjelmassasi on siis aikapommi. Varmista siis, että asetat tuollaisen osoittimen null-osoittimeksi, niin vaara katoaa.

Huom! Edellä kuvattua osoitinta kutsutaan joskus villiksi osoittimeksi.

const-osoittimet

Voit käyttää const-avainsanaa osoittimien yhteydessä ja sijoittaa const-sanan ennen tyyppiä, tyypin jälkeen tai molempiin paikkoihin. Seuraavassa on muutamia laillisia esittelyjä:

```
const int *pOne;
int *const pTwo;
const int *const pThree;
```

Uusi käsite pOne on osoitin vakioon kokonaislukuun. Osoitettua arvoa ei voida muuttaa tämän osoittimen avulla. Et voi siis kirjoittaa

```
*pOne = 5;
```

Jos yrität tehdä noin, kääntäjä antaa virheilmoituksen.

Uusi käsite pTwo on int-tyyppinen vakio-osoitin. Kokonaislukua voidaan muuttaa, mutta pTwo ei voi osoittaa minnekään muualle. Vakio-osoittimeen ei voida sijoittaa uutta arvoa.

Et voi siis kirjoittaa

```
pTwo = &x;
```

pThree on vakio-osoitin, joka osoittaa vakioon kokonaislukuun. Osoitettua arvoa ei voida muuttaa eikä osoittimeen voida sijoittaa uutta arvoa.

Vedä kuvitteellinen viiva asteriskin oikealle puolelle. Jos viivan vasemmalla puolella on const-sana, on kohde vakio. Jos const on viivan oikealla puolella, on osoitin itse vakio.

```
const int *p1;  //osoitettu int on vakio
int * const p2;  // p2 on vakio-osoitin, se ei voi osoittaa muualle
```

const-osoittimet ja -jäsenfunktiot

Luvussa 6 sait tietää, että jäsenfunktio voi olla const-tyyppinen. Kun funktio esitellään const-tyyppisenä, kääntäjä antaa virheilmoituksen, jos tuon funktion sisällä yritetään muuttaa kohteen tietoa.

Jos osoitin on const-tyyppinen, voidaan vain const-tyyppisiä metodeita kutsua tuota osoitinta käyttäen. Listaus 10.5 havainnollistaa edellä kerrottua.

Listaus 10.5. Osoittimien käyttäminen const-tyyppisten olioiden yhteydessä.

```
1:  // Listaus 10.5
2:  // const-metodit ja osoittimet
3:
4:  #include <iostream.h>
```

```
5:
6:  class Rectangle
7:  {
8:  public:
9:    Rectangle();
10:   ~Rectangle();
11:   void SetLength(int length) { itsLength = length; }
12:   int GetLength() const { return itsLength; }
13:
14:   void SetWidth(int width) { itsWidth = width; }
15:   int GetWidth() const { return itsWidth; }
16:
17: private:
18:   int itsLength;
19:   int itsWidth;
20: };
21:
22: Rectangle::Rectangle():
23: itsWidth(5),
24: itsLength(10)
25: {}
26:
27: Rectangle::~Rectangle()
28: {}
29:
30: int main()
31: {
32:   Rectangle* pRect =  new Rectangle;
33:   const Rectangle * pConstRect = new Rectangle;
34:   Rectangle * const pConstPtr = new Rectangle;
35:   cout << "pRect width: " << pRect->GetWidth() << " feet\n";
36:   cout << "pConstRect width: " ;
37:   cout << pConstRect->GetWidth() << " feet\n";
38:   cout << "pConstPtr width: " << pConstPtr->GetWidth() << " feet\n";
39:
40:   pRect->SetWidth(10);
41:   // pConstRect->SetWidth(10);
42:   pConstPtr->SetWidth(10);
43:
44:   cout << "pRect width: " << pRect->GetWidth() << " feet\n";
45:   cout << "pConstRect width: ";
46:   cout << pConstRect->GetWidth() << " feet\n";
47:   cout << "pConstPtr width: " << pConstPtr->GetWidth() << " feet\n";
48:   return 0;
49: }
```

Tulostus

```
pRect width:    5 feet
pConstRect width:   5 feet
pConstPtr width:    5 feet
pRect width:    10 feet
pConstRect width:   5 feet
pConstPtr width:    10 feet
```

Analyysi

Rivit 6-20 esittelevät Rectangle-luokan. Rivi 12 esittelee GetLength() metodin const-tyyppisenä. Rivi 32 esittelee osoittimen Rectangle-olioon. Rivi 33

esittelee osoittimen pConstRect, joka osoittaa vakioon Rectangle-olioon. Rivi 34 esittelee vakion osoittimen pConstPtr, joka osoittaa Rectangle-olioon.

Rivit 35-38 tulostavat leveysarvot.

Rivillä 40 käytetään pRect-osoitinta asettamaan suorakaiteen leveydeksi arvon 10. Rivillä 41 käytettäisiin pConstRect-osoitinta, mutta se on esitelty osoittamaan vakioon Rectangle-olioon. Siksi se ei voi kutsua ei-vakiota metodia ja se kommentoidaan pois. Rivillä 42 kutsutaan pConstPtr metodia SetWidth(10). Rivillä 34 esitellään pConstPtr vakio-osoittimena (osoittaa suorakaiteeseen). Toisin sanoen osoitin on vakio eikä voi osoittaa minnekään muualle. Suorakaide ei ole kuitenkaan vakio, joten tuo kutsu on laillinen.

const-tyyppiset this-osoittimet

Kun olio esitellään const-tyyppisenä, esitellään samalla myös this-osoitin osoittimena const-olioon. Tällaista const-tyyppistä this-osoitinta voidaan käyttää const-tyyppisten metodien kanssa.

Vakiot oliot ja osoittimet ovat esillä vielä seuraavassa luvussa, jossa käsitellään muun muassa viittauksia vakio-olioihin.

Yhteenveto

Osoittimet ovat tehokkaita työkaluja olioiden käsittelyyn vapaalla muistialueella. Ne aiheuttavat riskejä muistivuodon ja villien osoittimien muodossa. Mutta huolellisesti käytettyinä ne ovat turvallisia ja tehokkaita.

Osoittimet voidaan esitellä vakioina, jolloin hyödynnetään kääntäjää löytämään paikat, joissa osoittimia käytetään väärällä tavalla.

Kysymyksiä ja Vastauksia

K
Miksi kohde kannattaa esitellä const-tyyppisenä, jos se kerran rajoittaa sen käyttöä?
V
Ohjelmoijana haluat usein käyttää kääntäjää virheiden hakemisessa. Eräs yleinen ja vaarallinen virhetilanne syntyy, kun jokin funktio muuttaa objektia tavalla, joka ei ole ilmeinen kutsuvalle funktiolle. Kohteen esitteleminen const-tyyppisenä estää tällaiset muutokset.