Osa
III

12. oppitunti

Kehittyneet viittaukset ja osoittimet

Edelliset kolme lukua ovat käsitelleet viittausten ja osoittimien käyttöä. Tämän luvun aiheita ovat:

- ❏ Kuinka viittausten viemisellä voidaan tehostaa ohjelmia
- ❏ Kuinka päätetään, milloin käyttää osoittimia ja milloin viittauksia
- ❏ Kuinka vältetään muistiongelmat osoittimia käytettäessä
- ❏ Kuinka vältetään viittausongelmat

Viittauksena vieminen ohjelman tehostamiseksi

Aina, kun kohde viedään funktiolle arvona, tehdään kopio kohteesta. Ja aina, kun kohde palautetaan funktiosta arvona, tehdään myöskin kopio.

Käytettäessä laajoja, käyttäjien määrittelemiä kohteita, aiheuttaa kopiointi menetyksiä. Tällöin käytetään enemmän muistia kuin tarvitaan ja ohjelma on hitaampi.

Käyttäjän luoman oliokohteen koko pinossa on jäsenmuuttujien summa. Nuo jäsenmuuttujat voivat vuorostaan olla käyttäjän luomia olioita. Kun tuollainen massiivinen rakenne kopioidaan pinoon, on selvää, että suorituskyky huononee ja muistia tuhlaantuu runsaasti.

Myös muita kustannuksia aiheutuu. Luomiesi luokkien kohdalla luodaan kukin noista tilapäisistä kopioista kääntäjän kutsuessa erikoismuodostinta, kopiomuodostinta. Luvussa 20, "Erikoisluokat ja -funktiot", käsitellään kopiomuodostimia ja niiden kehittämistä. Tällä erää on hyvä tietää, että kopiomuodostinta kutsutaan joka kerta muodostettaessa olion kopio pinoon.

Kun tilapäiskopio tuhotaan funktion päättyessä, kutsutaan olion tuhoajafunktiota. Jos olio palautetaan arvona, on tehtävä kopio oliosta ja myös tuhottava se.

Laajojen kohteiden kohdalla voivat nuo muodostimen ja tuhoajafunktion kutsut vaikuttaa negatiivisesti suorituskykyyn ja muistin käyttöön. Tätä ajatusta havainnollistetaan listauksessa 12.1, jossa luodaan käyttäjän määrittelemä olio: SimpleCat. Todellinen olio olisi laajempi ja tuhlailevampi, mutta esimerkkimme riittää näyttämään, kuinka usein täytyy kopiomuodostinta ja tuhoajafunktiota kutsua.

Listaus 12.1 luo SimpleCat-olion ja kutsuu sitten kahta funktiota. Ensimmäinen funktio saa Cat-olion arvona ja palauttaa sen arvona. Toinen funktio vastaanottaa osoittimen olioon, eikä siis itse oliota ja se palauttaa myöskin osoittimen olioon.

Listaus 12.1. Olioiden vieminen viittauksina.

```
44: //Listaus 12.1
45: // olio-osoittimien vienti
46:
47: #include <iostream.h>
48:
49: class SimpleCat
50: {
51: public:
52:   SimpleCat ();                  // muodostin
53:   SimpleCat(SimpleCat&);     // kopiomuodostin
54:   ~SimpleCat();                  // tuhoaja
55: };
```

```
56:
57: SimpleCat::SimpleCat()
58: {
59:   cout << "Simple Cat Constructor...\n";
60: }
61:
62: SimpleCat::SimpleCat(SimpleCat&)
63: {
64:   cout << "Simple Cat Copy Constructor...\n";
65: }
66:
67: SimpleCat::~SimpleCat()
68: {
69:   cout << "Simple Cat Destructor...\n";
70: }
71:
72: SimpleCat FunctionOne (SimpleCat theCat);
73: SimpleCat* FunctionTwo (SimpleCat *theCat);
74:
75: int main()
76: {
77:   cout << "Making a cat...\n";
78:   SimpleCat Frisky;
79:   cout << "Calling FunctionOne...\n";
80:   FunctionOne(Frisky);
81:   cout << "Calling FunctionTwo...\n";
82:   FunctionTwo(&Frisky);
83:   return 0;
84: }
85:
86: // FunctionOne, viedään arvo
87: SimpleCat FunctionOne(SimpleCat theCat)
88: {
89:   cout << "Function One. Returning...\n";
90:   return theCat;
91: }
92:
93: // functionTwo, viedään viittaus
94: SimpleCat* FunctionTwo (SimpleCat  *theCat)
95: {
96:   cout << "Function Two. Returning...\n";
97:   return theCat;
98: }
```

Tulostus

```
1:  Making a cat…
2:  Simple Cat Constructor…
3:  Calling FunctionOne…
4:  Simple Cat Copy Constructor…
5:  Function One. Returning…
6:  Simple Cat Copy Constructor…
7:  Simple Cat Destructor…
8:  Simple Cat Destructor…
9:  Calling FunctionTwo…
10: Function Two. Returning…
11: Simple Cat Destructor…
```

Huom! Rivinumerot eivät tulostu, vaan ne on lisätty selitystekstiä varten.

Analyysi
Riveillä 6-12 esitellään yksinkertaistettu luokka SimpleCat. Muodostin, kopiomuodostin ja tuhoajafunktiot tulostavat kukin selittävän viestin, joka kertoo niiden kutsumisesta.

Rivillä 34 tulostaa main() viestin (tulostusrivi 19). Rivillä 35 luodaan SimpleCat-olio. Tällöin kutsutaan muodostinta ja muodostimen tulostus on rivillä 2.

Rivillä 36 kertoo main(), että se kutsuu funktiota FunctionOne(), joka luo 3. tulostusrivin. Koska FunctionOne() ottaa SimpleCat-olion arvona, luodaan paikallinen kopio oliosta pinoon. Tällöin tarvitaan kopiomuodostinta, joka luo 4. tulostusrivin.

Ohjelman suoritus siirtyy kutsutun funktion riville 45, joka tulostaa 5. tulostusrivin. Funktio päättyy palauttaen SimpleCat-olion arvona. Tällöin luodaan vielä yksi kopio oliosta kopiomuodostimen avulla (tulostusrivi 6).

Funktion FunctionOne() palauttamaa arvoa ei sijoiteta mihinkään, joten palautusta varten luotu tilapäisolio heitetään menemään. Tällöin tarvitaan tuhoajafunktiota, joka luo tulostusrivin 7. Koska FunctionOne() on päättynyt, sen paikallista kopiota ei enää voida käsitellä ja se tuhotaan tuhoajafunktion toimesta (tulostusrivi 8).

Ohjelman suoritus palaa main()-funktioon, jolloin kutsutaan funktiota FunctionTwo(), jolle parametrit viedään viittauksena. Mitään kopiota ei luoda, joten tulostustakaan ei synny. FunctionTwo() tulostaa viestin tulostusriville 10 ja palauttaa SimpleCat-olion viittauksena, eikä muodostimen ja tuhoajafunktion kutsuihin ole tarvetta.

Ohjelma päättyy ja Friskyn elinikä päättyy, jolloin on vielä kutsuttava tuhoajafunktiota (tulostusrivi 11).

Tuloksista näemme, että FunctionOne()-kutsu aiheutti kaksi kopiomuodostimen ja kaksi tuhoajafunktion kutsua, koska olio vietiin sille arvona. Sen sijaan FunctionTwo()-kutsu ei aiheuttanut yhtäkään kopiomuodostimen tai tuhoajafunktion kutsua.

const-osoittimen vieminen funktiolle

Vaikkakin osoittimien vienti funktiolle FunctionTwo() oli tehokas menettely, oli se myös vaarallinen tapa. FunctionTwo() ei voi muuttaa sille vietyä SimpleCat-oliota ja kuitenkin sille viedään olion osoite. Osoitteen vieminen tekee oliosta muutoksille alttiin ja tällöin menetetään se suojaus, joka arvona viemisellä taasen taataan.

Arvona vieminen on kuin taideteosten valokuvien antaminen museolle itse taideteosten sijaan. Jos valokuville tehdään ilkivaltaa, säilyvät alkuperäiset teokset kuitenkin koskemattomina. Viittauksena vieminen on kuin kotiosoitteen antaminen museolle ja vieraiden kutsuminen paikan päälle tutustumaan alkuperäisiin teoksiin. Vaarat ovat siis ilmeiset.

Ratkaisuna on vakio-osoittimen käyttö. Funktiolle vietävä osoitin on const-tyyppinen, SimpleCat-olioon osoittava osoitin. Tällöin olio suojataan muutoksilta. Listaus 12.2 esittelee tämän menettelyn.

Listaus 12.2. const-osoittimen vienti funktiolle.

```
1:  //Listaus 12.2
2:
3:  // const-osoittimet olioihin
4:
5:  #include <iostream.h>
6:
7:  class SimpleCat
8:  {
9:  public:
10:   SimpleCat();
11:   SimpleCat(SimpleCat&);
12:   ~SimpleCat();
13:
14:   int GetAge() const { return itsAge; }
15:   void SetAge(int age) { itsAge = age; }
16:
17: private:
18:   int itsAge;
19: };
20:
21: SimpleCat::SimpleCat()
22: {
23:   cout << "Simple Cat Constructor...\n";
24:   itsAge = 1;
25: }
26:
27: SimpleCat::SimpleCat(SimpleCat&)
28: {
29:   cout << "Simple Cat Copy Constructor...\n";
30: }
31:
32: SimpleCat::~SimpleCat()
33: {
34:   cout << "Simple Cat Destructor...\n";
35: }
36:
37: const SimpleCat * const
38: FunctionTwo (const SimpleCat * const theCat);
39:
40: int main()
41: {
42:   cout << "Making a cat...\n";
43:   SimpleCat Frisky;
44:   cout << "Frisky is ";
45:   cout << Frisky.GetAge() << " years old\n";
46:   int age = 5;
47:   Frisky.SetAge(age);
```

```
48:    cout << "Frisky is ";
49:    cout << Frisky.GetAge() << " years old\n";
50:    cout << "Calling FunctionTwo...\n";
51:    FunctionTwo(&Frisky);
52:    cout << "Frisky is ";
53:    cout << Frisky.GetAge() << " years old\n";
54:    return 0;
55: }
56:
57: // functionTwo, const-osoitin viedään
58: const SimpleCat * const
59: FunctionTwo (const SimpleCat * const theCat)
60: {
61:    cout << "Function Two. Returning...\n";
62:    cout << "Frisky is now " << theCat->GetAge();
63:    cout << " years old \n";
64:    // theCat->SetAge(8);   const!
65:    return theCat;
66: }
```

Tulostus

```
Making a cat…
Simple Cat Constructor…
Frisky is 1 years old
Frisky is 5 years old
Calling FunctionTwo…
Function Two. Returning…
Frisky is now 5 years old
Frisky is 5 years old
Simple Cat Destructor…
```

Analyysi

SimpleCat-luokkaan on lisätty kaksi funktiota: GetAge() ja SetAge(). GetAge() (rivi 14) on vakiofunktio, SetAge() (rivi 15) ei taas ole vakio. Luokkaan on myös lisätty jäsenmuuttuja itsAge riville 18.

Muodostin, kopiomuodostin ja tuhoajafunktio tulostavat edelleenkin viestinsä. Kopiomuodostinta ei koskaan kutsuta, koska olio viedään viittauksena, jolloin kopioita ei tehdä. Rivillä 43 luodaan olio, jonka oletusikä tulostetaan rivillä 45.

Rivillä 47 asetetaan muuttujaan itsAge arvo funktiolla SetAge ja vastaava tulostus on rivillä 49. Funktiota FunctionOne() ei käytetä ohjelmassa, mutta funktiota FunctionTwo() kutsutaan. Funktiota on muutettu hieman, parametri ja palautusarvo on nyt esitelty ottamaan ja palauttamaan vakio-osoitin vakio-olioon.

Koska parametri ja palautusarvo viedään edelleenkin viittauksena, ei kopioita tarvita eikä siis kopiomuodostinta kutsuta lainkaan. Funktion FunctionTwo() osoitin on kuitenkin nyt vakio eikä se siten voi kutsua ei-vakiota metodia SetAge(). Jos tuota kutsua (rivi 57) ei olisi kommentoitu pois, ei ohjelmaa olisi voitu kääntää.

Huomaa, että main()-funktion luoma olio ei ole vakio ja Frisky voi kutsua metodia SetAge(). Tuon ei-vakion olion osoite viedään funktiolle FunctionTwo(), mutta koska funktion esittelyssä osoittimen on määrä olla vakio, käsitellään oliota ikäänkuin se olisi vakio!

Vaihtoehtona viittaukset

Listaus 12.2 ratkaisee ylimääräisten kopioiden ongelman ja säästää ohjelman kopiomuodostimen ja tuhoajafunktion kutsuilta. Se käyttää vakio-osoittimia vakio-olioihin, jolloin olion muuttamisongelma tulee ratkaistuksi. Listaus on silti hieman sekava, koska funktiolle viedyt oliot ovat osoittimia.

Koska olio ei voi koskaan olla null, olisi helpompaa käyttää funktiota, jos sille vietäisiin osoittimen sijaan viittaus. Listaus 12.3 havainnollistaa tätä menettelyä.

Listaus 12.3. Olio-viittausten vieminen funktiolle.

```
1:  //Listaus 12.3
2:  // olioviittausten vienti
3:
4:  #include <iostream.h>
5:
6:  class SimpleCat
7:  {
8:  public:
9:    SimpleCat();
10:   SimpleCat(SimpleCat&);
11:   ~SimpleCat();
12:
13:   int GetAge() const { return itsAge; }
14:   void SetAge(int age) { itsAge = age; }
15:
16: private:
17:   int itsAge;
18: };
19:
20: SimpleCat::SimpleCat()
21: {
22:   cout << "Simple Cat Constructor...\n";
23:   itsAge = 1;
24: }
25:
26: SimpleCat::SimpleCat(SimpleCat&)
27: {
28:   cout << "Simple Cat Copy Constructor...\n";
29: }
30:
31: SimpleCat::~SimpleCat()
32: {
33:   cout << "Simple Cat Destructor...\n";
34: }
35:
36: const     SimpleCat & FunctionTwo (const SimpleCat & theCat);
37:
```

```
38: int main()
39: {
40:   cout << "Making a cat...\n";
41:   SimpleCat Frisky;
42:   cout << "Frisky is " << Frisky.GetAge() << " years old\n";
43:   int age = 5;
44:   Frisky.SetAge(age);
45:   cout << "Frisky is " << Frisky.GetAge() << " years old\n";
46:   cout << "Calling FunctionTwo...\n";
47:   FunctionTwo(Frisky);
48:   cout << "Frisky is " << Frisky.GetAge() << " years old\n";
49:   return 0;
50: }
51:
52: // functionTwo: viedään viittaus const-olioon
53: const SimpleCat & FunctionTwo (const SimpleCat & theCat)
54: {
55:   cout << "Function Two. Returning...\n";
56:   cout << "Frisky is now " << theCat.GetAge();
57:   cout << " years old \n";
58:   // theCat.SetAge(8);   const!
59:   return theCat;
60: }
```

Tulostus

```
Making a cat…
Simple Cat Constructor…
Frisky is 1 years old
Frisky is 5 years old
Calling FunctionTwo…
Function Two. Returning…
Frisky is now 5 years old
Frisky is 5 years old
Simple Cat Destructor…
```

Analyysi

Tulostus on samanlainen kuin listauksessa 12.2. Ainoa merkittävä ero on siinä, että FunctionTwo() ottaa ja palauttaa nyt viittauksen vakio-olioon. Toistettakoon vielä, että viittausten käyttö on yksinkertaisempaa kuin osoittimien käyttö ja viittauksilla saadaan aikaan samat säästöt ja teho sekä const-käytön tarjoama turvallisuus.

Milloin käyttää viittauksia ja milloin osoittimia

C++ -ohjelmoijat pitävät viittauksia paljon parempina kuin osoittimia. Viittaukset ovat selkeämpiä ja helpompia käyttää. Ne kätkevät hyvin informaation, kuten aiemmasta esimerkistä nähtiin.

Viittauksilla ei voida kuitenkaan uudelleen viitata. Jos on osoitettava ensin yhteen kohteeseen ja sitten myöhemmin toiseen, on käytettävä osoittimia.

Viittaus ei voi olla arvoltaan null, joten jos olion on saatava olla null jossain tilanteessa, ei viittausta voida käyttää. Tällöin on käytettävä osoitinta.

Esimerkki edellä mainitusta tilanteesta on new-operaattori. Jos new ei voi varata muistia, se palauttaa null-osoittimen. Koska viittaus ei voi olla null, et voi alustaa viittausta olemaan tuon muistialueen sijaisnimi ennen kuin olet tarkistanut, ettei se ole null. Seuraava esimerkki näyttää, kuinka tuollainen tilanne hoidetaan:

```
int *pInt = new int;
if (pInt != NULL)
int &rInt = *pInt;
```

Tässä esimerkissä esitellään int-osoitin, pInt ja alustetaan se new-operaattorin palauttamalla muistilla. Sitten pInt-osoittimen osoittama osoite tarkistetaan: jos se ei ole null, annetaan pInt-osoittimelle viittaus. Viittaus rInt on nyt sijaisnimi new-operaattorin palauttamalle int-oliolle.

Älä palauta viittausta olioon, joka ei ole näkyvyysalueella!

Kun C++ -ohjelmoijat oppivat viemään viittauksena, he villiintyvät ja voivat ylittää oikeutensa. Muistakaamme, että viittaus on aina sijaisnimi jollekin toiselle kohteelle. Jos viet tai palautat viittauksen, kysy itseltäsi "Mikä on se olio, jonka sijaisnimeä käytän ja onko se yhä olemassa aina käyttäessäni sitä?"

Listaus 12.4 havainnollistaa vaaratilannetta, jossa palautetaan viittaus olioon, jota ei enää ole olemassa.

Listaus 12.4. Viittauksen palauttaminen kohteeseen, jota ei enää ole olemassa.

```
1:  // Listaus 12.4
2:  // Palauttaa viittauksen olioon,
3:  // jota ei enää ole
4:
5:  #include <iostream.h>
6:
7:  class SimpleCat
8:  {
9:  public:
10:   SimpleCat (int age, int weight);
11:   ~SimpleCat() {}
12:   int GetAge() { return itsAge; }
13:   int GetWeight() { return itsWeight; }
14: private:
15:   int itsAge;
16:   int itsWeight;
17: };
```

```
18:
19: SimpleCat::SimpleCat(int age, int weight):
20: itsAge(age), itsWeight(weight) {}
21:
22: SimpleCat &TheFunction();
23:
24: int main()
25: {
26:   SimpleCat &rCat = TheFunction();
27:   int age = rCat.GetAge();
28:   cout << "rCat is " << age << " years old!\n";
29:   return 0;
30: }
31:
32: SimpleCat &TheFunction()  // palauttaa viittauksen
33: {
34:   SimpleCat Frisky(5,9);  // Paikallinen olio luodaan !!!
35:   return Frisky;
36: }
```

Tulostus

```
Compile error: Attempting to return a reference to a local object!
```

Varoitus! Tätä ohjelmaa ei voida kääntää Borlandin kääntäjällä. Sen sijaan se voidaan kääntää Microsoftin kääntäjällä; tämä on kuitenkin huonoa koodia.

Analyysi

Riveillä 7-17 esitellään SimpleCat-luokka. Rivillä 26 alustetaan viittaus SimpleCat-luokkaan kutsumalla funktiota TheFunction, joka esitellään rivillä 22 palauttamaan SimpleCat-viittaus.

Funktion TheFunction() rungossa esitellään paikallinen SimpleCat-olio ja samalla alustetaan sen ikä ja paino. Funktio palauttaa viittauksen paikalliseen olioon. Jotkut kääntäjät pystyvät havaitsemaan tämän virheen eivätkä salli ohjelman ajamista. Jotkut kääntäjät taas sallivat ajamisen, mikä voi johtaa odottamattomiin tuloksiin.

Kun TheFunction päättyy, tuhotaan paikallinen Frisky-olio (kivuttomasti, totta kai). Funktion palauttama viittaus on sijaisnimi olemattomaan olioon, mikä on siis huonoa koodia.

Olioviittauksen palauttaminen kekoon

Haluaisit varmaan yrittää ratkaista listauksen 12.4 ongelman sallimalla funktion TheFunction() luoda Frisky-olion kekoon. Sillä tavoin olisi Frisky vielä olemassa funktion päättymisen jälkeen.

Tämän lähestymistavan ongelma on seuraavanlainen: mitä tehdään Friskylle varatulle muistille, kun sitä ei enää tarvita? Listaus 12.5 havainnollistaa tätä ongelmaa.

Listaus 12.5. Muistikato.

```
1:  // Listaus 12.5
2:  // Muistikatoa
3:
4:  #include <iostream.h>
5:
6:  class SimpleCat
7:  {
8:  public:
9:    SimpleCat (int age, int weight);
10:   ~SimpleCat() {}
11:   int GetAge() { return itsAge; }
12:   int GetWeight() { return itsWeight; }
13:
14: private:
15:   int itsAge;
16:   int itsWeight;
17: };
18:
19: SimpleCat::SimpleCat(int age, int weight):
20: itsAge(age), itsWeight(weight) {}
21: SimpleCat & TheFunction();
22:
23: int main()
24: {
25:   SimpleCat & rCat = TheFunction();
26:   int age = rCat.GetAge();
27:   cout << "rCat is " << age << " years old!\n";
28:   cout << "&rCat: " << &rCat << endl;
29:   // Kuinka muistista eroon?
30:   SimpleCat * pCat = &rCat;
31:   delete pCat;
32: // Mihin rCat nyt viittaa?
33:   return 0;
34: }
35:
36: SimpleCat &TheFunction()
37: {
38:   SimpleCat * pFrisky = new SimpleCat(5,9);  // Olio kekoon!!!
39:   cout << "pFrisky: " << pFrisky << endl;
40:   return *pFrisky;
41: }
```

Tulostus

```
pFrisky:  0x2bf4
rCat is 5 years old
&rCat:   0x2bf4
```

Varoitus! Ohjelma käännetään ja linkitetään ja se näyttää toimivan. Mutta on vain ajan kysymys, milloin se kaatuu.

Analyysi
Listauksen TheFunction()-funktiota on muutettu siten, ettei se enää palauta viittausta paikalliseen muuttujaan. Muistia varataan vapaasta tilasta ja siihen kytketään osoitin rivillä 37. Osoittimen tallentama osoite tulostetaan ja sen jälkeen osoittimella viitataan uudelleen ja SimpleCat-olio palautetaan viittauksena.

Rivillä 25 sijoitetaan funktion TheFunction palauttama arvo SimpleCat-viittaukseen ja tuota oliota käytetään hakemaan kissan ikä, joka tulostetaan rivillä 27.

Todistaaksemme, että main()-funktiossa esitelty viittaus viittaa olioon, joka on sijoitettu vapaaseen muistitilaan funktion TheFunction() sisällä, käytetään rCat-viittauksen yhteydessä osoiteoperaattoria. Siinä onkin sen olion osoite, johon se viittaa ja tuo osoite on sama kuin vapaassa tilassa olevan olion osoite.

Varoitus! Tähän mennessä on kaikki hyvin. Mutta kuinka tuo muistialue vapautetaan? Delete-operaattoria ei voida käyttää tuolle viittaukselle. Eräs näppärä ratkaisu on luoda toinen osoitin ja alustaa se rCat-viittauksen osoitteella. Se ei tuhoa muistia ja tukkii muistivuodon. Eräs pikkuongelma kuitenkin vielä on: Mihin rCat viittaa rivin 31 jälkeen? Kuten aiemmin mainittiin, viittauksen tulee aina olla sijaisnimi todelliseen olioon; jos se viittaa null-kohteeseen (kuten nyt), ohjelma on virheellinen.

Varoitus! Vaikka ohjelma, jossa on viittaus null-kohteeseen, käännettäisiinkin, on ohjelma virheellinen ja sen toiminta on odottamatonta.

Tähän ongelmaan on todellisuudessa kolme ratkaisutapaa. Ensinnäkin voimme esitellä SimpleCat-olion rivillä 25 ja palauttaa sen funktiosta TheFunction arvona. Toiseksi voimme esitellä SimpleCat-olion vapaaseen tilaan funktiossa TheFunction(), mutta annamme funktion palauttaa osoittimen tuohon muistiin. Sen jälkeen voi kutsuva funktio tuhota osoittimen, kun sitä ei enää tarvita.

Kolmantena ratkaisuna ja oikeana sellaisena on olion esittely kutsuvassa funktiossa, jolloin se voidaan viedä funktiolle TheFunction() viittauksena.

Kuka omistaa osoittimen?

Kun ohjelma varaa muistia vapaasta tilasta, palautetaan osoitin. On tärkeää, että tuohon muistiin osoittavaa osoitinta ei kadoteta, koska muutoin muistia ei voida vapauttaa ja syntyy muistikatoa.

Kun tuota muistialuetta siirretään funktiolta toiselle, joku aina omistaa osoittimen. Yleensä muistilohkon arvo viedään viittauksena ja se funktio, joka loi muistilohkon, myös tuhoaa sen. Mutta tämä on vain yleissääntö.

On vaarallista, jos yksi funktio luo muistin ja toinen funktio taas tuhoaa sen. Tietämättömyys siitä, kuka omistaa osoittimen, voi johtaa yhteen kahdesta eri ongelmasta: unohdetaan tuhota osoitin tai osoitin tuhotaan kaksi kertaa. Kumpikin toiminto voi johtaa vakaviin ohjelmaongelmiin. On turvallisempaa muodostaa funktiot niin, että ne myös vapauttavat varaamansa muistin.

Jos funktiosi tulee varata muistia ja viedä se sitten kutsuvalle funktiolle, mieti, voisitko muuttaa käyttöliittymääsi. Anna kutsuvan funktion varata muistia ja vie se sitten muille funktioille viittauksina. Näin muistinhallinta pysyy yhden funktion käsissä.

Tee/Älä tee

Vie parametrit arvoina silloin, kun niin on tehtävä.
Anna palautusten tapahtua arvoina silloin, kun niin on tehtävä.
Älä vie parametreja viittauksina, jos viitattu kohde voi siirtyä pois näkyvyysalueelta.
Älä käytä viittauksia null-kohteisiin.

Yhteenveto

Tässä luvussa todettiin, että kohteiden vieminen funktioille viittauksina voi olla tehokkaampaa kuin niiden vieminen arvoina. Viittauksena vieminen mahdollistaa myös argumenttien arvojen muuttamisen kutsutun funktion sisällä ja muutettujen arvojen palauttamisen kutsuvalle funktiolle.

Funktioiden argumentit ja palautusarvot voidaan viedä viittauksina, mikä voidaan toteuttaa osoittimin tai viittauksin.

Luvussa käsiteltiin myös const-tyyppisiä osoittimia ja viittauksia, joiden avulla voidaan turvallisesti siirtää arvoja funktioiden välillä ja samalla säilytetään se tehokkuus, jonka viittauksena vieminen tuo ohjelmaan.

Kysymyksiä ja Vastauksia

K
Miksi tarvitaan osoittimia, jos viittauksia on helpompi käyttää?
V
Viittaukset eivät voi kohdistua null-kohteisiin eikä niitä voida asettaa viittaamaan toisiin kohteisiin. Osoittimet ovat joustavampia, mutta hieman vaikeampia käyttää.

K
Miksi funktion tulisi koskaan palauttaa arvonsa arvona?
V
Jos palautettava arvo on paikallinen, se täytyy palauttaa arvona tai muutoin palautetaan viittaus kohteeseen, jota ei ole olemassa.

K
Jos viittauksena palauttaminen kerran on vaarallista, miksei aina hoideta palauttamista arvona?
V
Viittauksena palauttaminen lisää joustavuutta. Muistia säästetään ja ohjelma toimii nopeammin.