Osa III

Muistin hallinta

Oppitunnit

Osa
III

9. oppitunti

Osoittimet

Tehokkaimpia C++ -työkaluja ovat mahdollisuudet, joilla voidaan käsitellä tietokoneen muistia suoraan osoittimia käyttäen. Osoittimia pidetään kuitenkin kaikkein vaikeaselkoisimpana C++ -ominaisuutena.

Mielestäni osoittimet ja niiden merkitys voidaan ymmärtää helposti, kunhan käytämme niiden tutkimiseen hieman aikaa. Tässä luvussa käsittelemmekin seuraavia aiheita:

- ❏ Mitä osoittimet ovat?
- ❏ Kuinka osoittimia esitellään ja käytetään?
- ❏ Mitä on vapaa muistialue ja kuinka muistia käsitellään?

Tässä luvussa selitetään osoittimien toiminta vaihe vaiheelta. Huomaa kuitenkin, että ymmärrät osoittimien tarpeellisuuden vasta kirjan edetessä pitemmälle.

Mikä on osoitin?

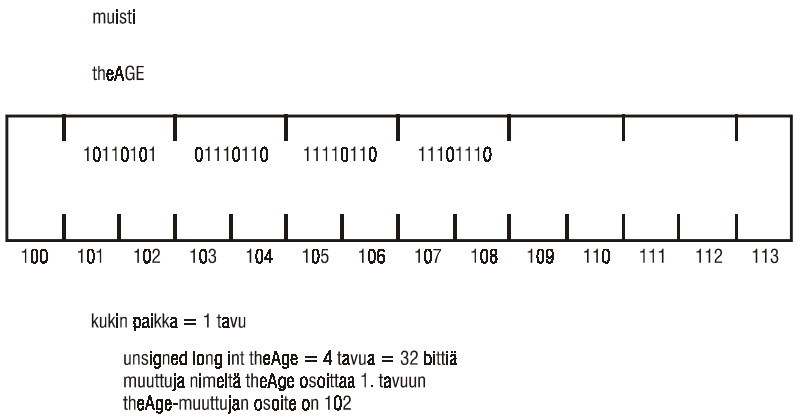
Uusi käsite Osoitin on muuttuja, johon tallennetaan muistiosoite.

Pysähdy. Lue edellä oleva lause uudelleen. Sinähän tiedät, mikä on muuttuja; se on kohde, johon voidaan tallentaa arvoja. Kokonaislukumuuttujaan voidaan tallentaa luku. Merkkimuuttujaan voidaan tallentaa kirjain. Osoitin on muuttuja, johon voidaan tallentaa muistiosoite.

Okei, mutta mikä on sitten muistiosoite? Tutkikaamme ensin tietokoneen muistia. Älä säikähdä, se ei ole mitään vaikeaa.

Tietokoneen muisti on paikka, johon arvot tallennetaan. Voimme ajatella, että muisti on jaettu numerojärjestyksessä oleviin muistipaikkoihin. Kukin näistä muistipaikoista on muistiosoite.

Jokaisen muuttujan jokainen arvo sijaitsee ainutkertaisessa sijaintipaikassa tietyssä osoitteessa. Kuvassa 9.1 on graafinen esitys, jossa muistiin on tallennettu unsigned long -tyyppinen muuttuja, theAge.



Kuva 9.1. Graafinen esitys theAge-muuttujan sijoittamisesta muistiin.

Eri tietokoneissa numeroidaan muisti erilaisin, monimutkaisin menettelyin. Yleensä ohjelmoijan ei tarvitse tietää minkään muuttujan tarkkaa osoitetta, koska tietokone hoitaa yksityiskohdat. Jos haluat saada tietoon nuo tiedot, voit käyttää osoiteoperaattoria (&), jonka käyttöä havainnollistetaan listauksessa 9.1.

Listaus 9.1. Muuttujien osoitteiden esittely.

```
1:  // Listaus 9.1 Esittelee osoiteoperaattorin
2:  //ja paikallisten muuttujien osoitteet
3:
4:  #include <iostream.h>
```

```
5:  int main()
6:  {
7:    unsigned short shortVar=5;
8:    unsigned long  longVar=65535;
9:    long sVar = -65535;
10:
11:   cout << "shortVar:\t" << shortVar;
12:   cout << "Address of shortVar:\t" <<  &shortVar << "\n";
13:   cout << "longVar:\t"  << longVar;
14:   cout << " Address of longVar:\t"  <<  &longVar  << "\n";
15:   cout << "sVar:\t" << sVar  << " Address of sVar:\t" <<  &sVar << "\n";
16:
17:   return 0;
18: }
```

Tulostus

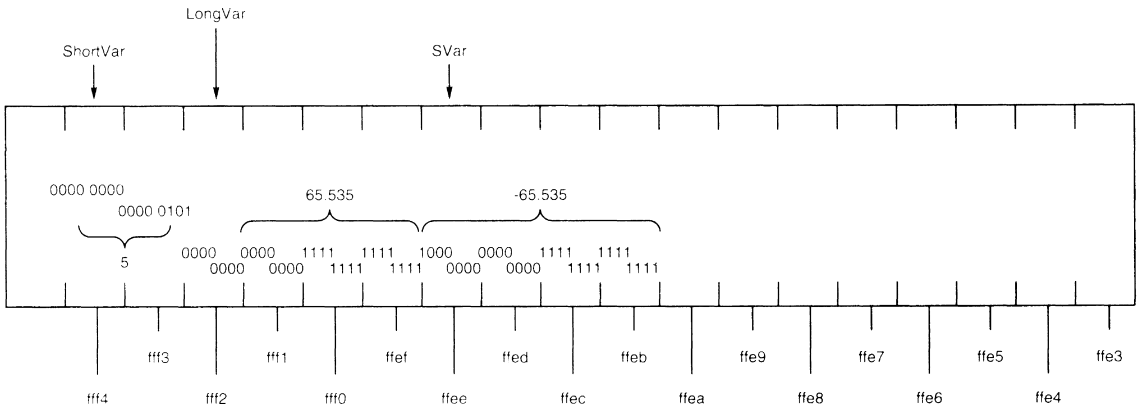
```
Shortvar: 5     Address of shortvar: 0x8fc9:fff4
Longvar: 65535  Address of longvar:  0x8fc9:fff2
SVar:      -65535   Address of sVar:      0x8fc9:ffee
```

(Tulostuksesi saattaa näyttää erilaiselta.)

Analyysi

Ohjelmassa esitellään ja alustetaan kolme muuttujaa: short-muuttuja rivillä 7, unsigned long -muuttuja rivillä 8 ja long-muuttuja rivillä 9. Niiden arvot ja osoitteet tulostetaan riveillä 11-15 osoiteoperaattorin (&) avulla.

Muuttujan shortVar arvo on 5 (kuten pitikin) ja sen osoite on 0x8fc9:fff4 käyttämässäni 80386-pohjaisessa koneessa. Tuo monimutkainen osoite riippuu käytetystä tietokoneesta ja saattaa vaihdella hieman joka kerta ohjelmaa ajettaessa. Omat tuloksesi saattavat olla erilaiset. Se, mikä ei muutu, on kahden ensimmäisen osoitteen erotus, joka on 2 tavua, mikäli koneesi käyttää kahden tavun long-kokonaislukuja. Toisen ja kolmannen osoitteen ero on 4 tavua. Kuva 9.2 havainnollistaa sitä, kuinka tämän ohjelman muuttujat tallennetaan muistiin.



Kuva 9.2. Muuttujien tallentamisen havainnollistaminen.

Ei ole mitään syytä tietää jokaisen muuttujan todellista muistiosoitetta. Sinun on vain tiedettävä, että jokaisella muuttujalla on osoite ja että muuttujaa varten tulee varata tarpeellinen määrä muistia.

Kuinka kääntäjä tietää, kuinka paljon muistia kukin muuttuja tarvitsee? Sen kertoo juuri muuttujan tietotyyppi.

Esiteltäessä esimerkiksi unsigned long -tyyppinen muuttuja, kääntäjä osaa varata muistia 4 tavua, koska kyseinen tyyppi on neljän tavun kokoinen. Kääntäjä huolehtii todellisen muistiosoitteen yhdistämisestä muuttujalle.

Osoitteen tallentaminen osoittimeen

Jokaisella muuttujalla on osoite. Voit tallentaa muuttujan osoitteen osoittimeen tietämättä tuota osoitetta.

Olettakaamme esimerkin vuoksi, että howOld on kokonaisluku. Osoitin nimeltä pAge, johon howOld-muuttujan osoite tallennetaan, voidaan esitellä seuraavasti:

```
int *pAge = 0;
```

Koodi esittelee osoittimen pAge, joka osoittaa int-tyyppiseen muuttujaan. Täten pAge on esitelty tallentamaan int-muuttujien osoitteita.

Huomaa, että pAge on samanlainen muuttuja kuin muutkin tavalliset muuttujat. Kun esittelet kokonaislukumuuttujan (int-tyyppisen muuttujan), se on asetettu tallentamaan kokonaislukuja. Kun esittelet osoitinmuuttujan (kuten pAge), se on asetettu tallentamaan osoitteita. Osoitin on siis vain erikoistyyppinen muuttuja, johon tallennetaan eri kohteiden muistiosoitteita. Esimerkissämme esiteltiin osoitinmuuttuja pAge tallentamaan kokonaislukumuuttujien osoitteita.

Esimerkissämme pAge alustettiin arvolla 0. Osoitin, jonka arvo on 0, on nimeltään null-osoitin. Kaikki luotavat osoittimet tulee alustaa osoittamaan johonkin. Jos et tiedä, mikä osoite muuttujaan sijoitetaan, laita se osoittamaan arvoon 0. Alustamatonta osoitinta kutsutaan villiksi osoittimeksi. Villi osoitin voi olla hyvin vaarallinen.

Huom! Muista aina: alusta osoittimesi!

Jos alustat pAge-osoittimen nollalla, sinun on tallennettava howOld-muuttujan osoite myöhemmin siihen. Voit toteuttaa sen seuraavasti:

```
unsigned short int howOld = 50;  // muuttujan määrittely
unsigned short int *pAge = 0;     // osoittimen määrittely
pAge=&howOld;//howOld-osoitteen tallentaminen pAge-osoittimeen
```

Ensimmäisellä rivillä luodaan muuttuja howOld - sen tyyppi on unsigned short int - ja alustetaan se arvolla 50. Toisella rivillä esitellään pAge osoittimena tyyppiin unsigned short int ja alustetaan se arvolla 0. Tunnistat pAge-muuttujan osoittimeksi asteriski (*) -merkistä, joka sijoitetaan tietotyypin ja nimen väliin.

Kolmas rivi sijoittaa muuttujan howOld osoitteen osoittimeen pAge. Osoitteen tallentaminen tunnistetaan osoiteoperaattorista (&). Jos osoiteoperaattoria ei käytetä, sijoitettaisiin pAge-muuttujaan muuttujan howOld arvo. Kyseessä ei ehkä olisi ollut sopiva osoite.

Nyt pAge-muuttujassa on muuttujan howOld osoite ja muuttujassa howOld on taas arvo 50. Sama olisi voitu toteuttaa lyhyemmin seuraavasti:

```
unsigned short int howOld = 50; // muuttujan määrittely
unsigned short int *pAge = &howOld;// osoittimen määrittely,
osoittaa howOld-muuttujaan
```

Muuttuja pAge on osoitin, joka sisältää nyt muuttujan howOld osoitteen. Voimme nyt käsitellä howOld-muuttujan arvoa pAge-osoittimen avulla. Muuttujan howOld käsittelyä osoittimen pAge avulla kutsutaan epäsuoraksi menettelyksi, koska siinä käsitellään howOld-muuttujaa epäsuorasti pAge-osoittimen kautta. Käsittelemme tätä menettelyä myöhemmin tässä luvussa.

Epäsuora menettely tarkoittaa sen arvon käsittelyä, joka sijaitsee osoittimen tallentamassa osoitteessa. Osoitin tarjoaa epäsuoran tavan käsitellä osoitteessa olevaa arvoa.

Osoittimien nimet

Osoittimet nimetään samoin perustein kuin tavallisetkin muuttujat. Tässä kirjassa käytetään tapaa, jossa osoittimen nimen eteen laitetaan kirjain p, kuten esimerkiksi pAge ja pNumber.

Epäsuoruusoperaattori

Epäsuoruusoperaattoria (*) sanotaan myös uudelleenviittausoperaattoriksi. Kun osoittimella uudelleenviitataan, viitataan osoittimella arvoon, joka on siinä osoitteessa, johon osoitin viittaa.

Tavalliset muuttujat sallivat niiden omien arvojen käsittelyn. Jos luot uuden muuttujan, joka on tyyppiä unsigned short int ja jonka nimi on yourAge ja haluat sijoittaa siihen muuttujassa howOld olevan arvon, voit kirjoittaa:

```
unsigned short int yourAge;
yourAge = howOld;
```

Osoitin mahdollistaa epäsuoran pääsyn sen muuttujan arvoon, jonka osoite siihen on tallennettu. Muuttujan howOld arvo voidaan sijoittaa uuteen muuttujaan yourAge osoittimen pAge avulla seuraavasti:

```
unsigned short int yourAge;
yourAge = *pAge;
```

Epäsuoruusoperaattori (*) osoittimen pAge edessä tarkoittaa "osoitteeseen tallennettua arvoa". Sijoituslause kertoo, että "hae pAge-osoitteessa oleva arvo ja sijoita se muuttujaan yourAge".

Huom! Epäsuoruusoperaattoria käytetään kahdella eri tavalla: esittelyyn ja uudelleenviittaukseen. Kun osoitin esitellään, tähti osoittaa, että kyseessä on osoitin, ei siis normaali muuttuja. Esimerkiksi:

```
unsigned short * Page = 0; //esitellään osoitin, joka on tyyppiä unsigned short
```

Kun osoittimella viitataan uudelleen, tähti kertoo, että nyt käsitellään osoittimen osoittamassa osoitteessa olevaa arvoa, ei siis osoitetta itseään:

```
*pAge = 5; // sijoitetaan 5 pAge-osoitteeseen
```

Huomaa myös, että asteriskia käytetään myös kertolaskuoperaattorina. Koodiyhteydestä kääntäjä osaa päätellä, mihin tarkoitukseen asteriskia tulee kulloinkin käyttää.

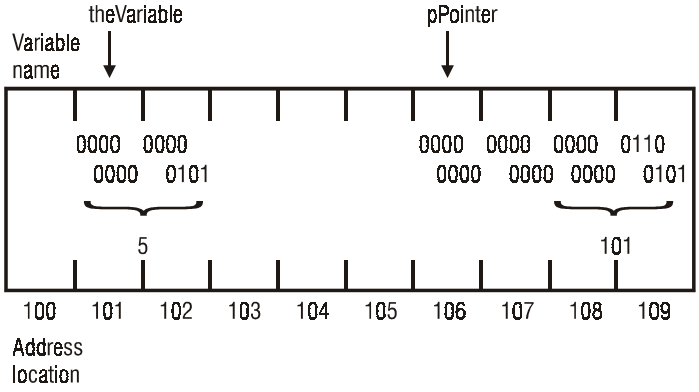
Osoittimet, osoitteet ja muuttujat

On tärkeää erottaa toisistaan osoitin, osoite, johon osoitin viittaa sekä arvo, joka osoittimessa olevaan osoitteeseen on tallennettu. Juuri tämä aiheuttaa epäselvyyttä osoittimien kohdalla.

Katso seuraavaa koodia:

```
int theVariable = 5;
int *pPointer = &theVariable;
```

Ensin esitellään muuttuja theVariable, joka alustetaan arvolla 5. Osoitin pPointer esitellään osoittamaan int-tyyppisiin muuttujiin ja se alustetaan muuttujan theVariable osoitteella. Osoitteessa, johon pPointer osoittaa on nyt siis arvo 5. Kuvassa 9.3 on graafinen esitys muuttujista theVariable ja pPointer.



Kuva 9.3. Kaaviokuva muistista.

Tiedon käsittely osoittimilla

Kun osoittimeen on sijoitettu muuttujan osoite, voit käyttää osoitinta muuttujassa olevan arvon käsittelyyn. Listaus 9.2 näyttää, kuinka paikallisen muuttujan osoite sijoitetaan osoittimeen ja kuinka muuttujassa olevaa arvoa sitten käsitellään osoittimen avulla.

Listaus 9.2. Tiedon käsittely osoittimien avulla.

```
1:  // Listaus 9.2 Osoittimien käyttö
2:
3:  #include <iostream.h>
4:
5:  typedef unsigned short int USHORT;
6:  int main()
7:  {
8:    USHORT myAge;         // muuttuja
9:    USHORT * pAge = 0;    // osoitin
10:   myAge = 5;
11:   cout << "myAge: " << myAge << "\n";
12:
13:   pAge = &myAge;     // sijoita myAge:n osoite pAge-osoittimeen
14:
15:   cout << "*pAge: " << *pAge << "\n\n";
16:
17:   cout << "*pAge = 7\n";
18:
19:   *pAge = 7;         // sijoita myAge-muuttujaan 7
20:
21:   cout << "*pAge: " << *pAge << "\n";
22:   cout << "myAge: " << myAge << "\n\n";
23:
24:
25:   cout << "myAge = 9\n";
26:
27:   myAge = 9;
28:
29:   cout << "myAge: " << myAge << "\n";
30:   cout << "*pAge: " << *pAge << "\n";
31:
32:   return 0;
33: }
```

Tulostus

```
myAge: 5
*pAge:  5

*pAge =  5
*pAge:  7
myAge: 7

myAge = 9
myAge: 9
*pAge: 9
```

Analyysi

Ohjelmassa esitellään kaksi muuttujaa: unsigned short -tyyppinen myAge ja osoitin nimeltä pAge, joka osoittaa tyyppiin unsigned short. Osoittimeen tallennetaan muuttujan myAge osoite. Rivillä 10 sijoitetaan arvo 5 muuttujaan myAge; se todennetaan rivin 11 tulostuksella.

Rivillä 13 sijoitetaan muuttujan myAge osoite osoittimeen pAge. Rivillä 15 käytetään osoitinta pAge uudelleenviittaukseen ja tehdään tulostus, joka osoittaa, että osoittimen pAge osoitteessa on muuttujaan myAge tallennettu arvo, 5. Rivillä 19 sijoitetaan arvo 7 osoittimessa pAge olevaan osoitteeseen. Tällöin myAge saa arvon 7 ja rivien 21-22 tulostus vahvistaa tämän.

Rivillä 27 sijoitetaan muuttujaan myAge arvo 9. Tämä arvo saadaan esille suoraan ja epäsuorasti osoittimen pAge uudelleenviittauksella rivillä 30.

Osoitteen tutkiminen

Osoittimet mahdollistavat osoitteiden käsittelyn, vaikkei osoitteiden todellisia arvoja tiedettäisikään. Tämän luvun jälkeen sinulle pitäisi olla täysin selvää, että sijoittaessasi muuttujan osoitteen osoittimeen, osoittimen tallentamana arvona on nimenomaan tuo osoite. Miksi emme tutkisi kuitenkin, että näin on? Listaus 9.3 antaa sinulle vahvistuksen asiaan.

Listaus 9.3. Osoittimeen tallennetun arvon tutkiminen.

```
1:  // Listaus 9.3 Mitä osoittimeen on tallennettu.
2:
3:  #include <iostream.h>
4:
5:  typedef unsigned short int USHORT;
6:  int main()
7:  {
8:    unsigned short int myAge = 5, yourAge = 10;
9:    unsigned short int * pAge = &myAge;  // osoitin
10:
11:   cout << "myAge:\t" << myAge <<  "\tyourAge:\t" << yourAge << "\n";
12:   cout << "&myAge:\t" << &myAge <<  "\t&yourAge:\t" << &yourAge <<"\n";
13:
14:   cout << "pAge:\t" << pAge << "\n";
15:   cout << "*pAge:\t" << *pAge << "\n";
```

```
16:
17:    pAge = &yourAge;       // uusi sijoitus
18:
19:    cout << "myAge:\t" << myAge <<  "\tyourAge:\t" << yourAge << "\n";
20:    cout << "&myAge:\t" << &myAge <<  "\t&yourAge:\t" << &yourAge <<"\n";
21:
22:    cout << "pAge:\t" << pAge << "\n";
23:    cout << "*pAge:\t" << *pAge << "\n";
24:
25:    cout << "&pAge:\t" << &pAge << "\n";
26:    return 0;
27: }
```

Tulostus

```
myAge:  5   yourAge:    10
&myAge: 0x355C  &yourAge:       0x355E
pAge:   0x355C
*pAge:  5
myAge:  5   yourAge:    10
&myAge: 0x355C  &yourAge:       0x355E
pAge:   0x355E
*pAge:  10
&pAge:  0x355A
```

(Tulostuksesi voi olla erilainen.)

Analyysi

Rivillä 8 esitellään tyyppiä unsigned short int olevat muuttujat myAge sekä yourAge. Rivillä 9 esitellään tyyppiä unsigned short int oleva osoitin nimeltä pAge. Se alustetaan muuttujan myAge osoitteella.

Riveillä 11-12 tulostetaan muuttujien myAge ja YourAge arvot. Rivi 14 tulostaa pAge-osoittimen sisällön eli muuttujan myAge osoitteen. Rivi 15 tulostaa arvon, joka saadaan osoittimen pAge uudelleenviittauksella eli muuttujan myAge arvon, 5.

Juuri tämä on olennaista osoittimissa. Rivi 14 kertoo, että osoittimessa pAge on muuttujan myAge arvo ja rivi 15 taas, että muuttujan myAge arvoa voidaan käsitellä osoittimen pAge uudelleenviittauksella. Varmista, että ymmärrät tämän täysin ennen jatkamista. Tutki koodia ja katso tulostusta.

Rivillä 17 sijoitetaan osoittimeen muuttujan yourAge osoite. Sen jälkeen tulostetaan arvot ja osoitteet uudelleen. Tulostus osoittaa, että osoittimessa pAge on nyt muuttujan yourAge osoite ja uudelleenviittaus mahdollistaa pääsyn muuttujan yourAge arvoon.

Rivi 25 tulostaa osoittimen pAge osoitteen. Silläkin on tietysti osoitteensa, joka voidaan tallentaa osoittimeen. (Osoittimen osoitteen sijoittaminen toiseen osoittimeen otetaan esille tuonnempana.)

Tee/Älä tee

Käytä epäsuoruusoperaattoria (*) osoittimessa olevaan osoitteeseen tallennetun tiedon käsittelyyn.

Alusta osoittimet joko jollakin osoitteella tai arvolla null (0).
Muista ero osoittimessa olevan osoitteen ja osoitteessa olevan arvon välillä.

Miksi käyttäisit osoittimia?

Olemme nyt käsitelleet vaiheittain muuttujan osoitteen tallentamisen osoittimeen. Käytännössä näin ei kuitenkaan koskaan tehdä. Voit ihmetellä, miksi osoittimia tuli sitten käyttää, koska meillä jo on pääsy muuttujan arvoon suoraan. Ainoa syy tämän tapaiseen osoittimien käyttöön oli esitellä osoittimien toimintaa. Nyt, kun tunnet osoittimet, voit ottaa ne todelliseen käyttöön. Osoittimia käytetään useimmiten seuraavaan kolmeen tehtävään:

- ❏ Muistissa olevan tiedon käsittelyyn
- ❏ Luokan jäsentiedon ja funktioiden käsittelyyn
- ❏ Muuttujien viemiseen viittauksina funktioille

Tämän luvun loppuosassa tutustutaan muistissa olevan tiedon hallintaan sekä luokan jäsentiedon ja -funktioiden käsittelyyn. Luvussa 10, "Kehittyneet osoittimet", opit muuttujien viemisen viittauksina.

Pino ja vapaa muistialue

Ohjelmoijat käyttävät yleensä viittä erilaista muistialuetta:

- ❏ Globaali nimiavaruus
- ❏ Vapaa muistialue
- ❏ Rekisterit
- ❏ Koodiavaruus
- ❏ Pino

Paikalliset muuttujat tallennetaan pinoon samoin kuin funktioiden parametrit. Koodi on tietenkin koodiavaruudessa ja globaalit muuttujat globaalissa nimiavaruudessa. Rekistereitä käytetään sisäisiin hallintafunktioihin kuten pinon huipun ja komento-osoittimen seurantaan. Lähes kaikki muu muisti annetaan vapaaksi muistiksi, jota kutsutaan joskus keoksi.

Paikallisten muuttujien ongelmana on niiden tilapäisyys: kun funktio päättyy, heitetään paikalliset muuttujat menemään. Globaalit muuttujat ratkaisevat tuon ongelman. Niiden ongelmana on kuitenkin se, että niihin

päästään käsiksi rajoittamattomasti joka kohdasta ohjelmaa, mikä johtaa vaikealukuisiin ja huonosti ylläpidettäviin ohjelmiin. Tiedon sijoittaminen vapaaseen muistiin ratkaisee nämä ongelmat.

Voit ajatella vapaata muistia suurena muistikennostona, jossa on tuhansia numerojärjestyksessä olevia paikkoja odottamassa tietojasi. Et voi kuitenkaan otsikoida noita paikkoja, kuten pinon kohdalla. Sinun on kysyttävä kunkin tarvitsemasi paikan osoitetta ja sijoitettava sitten osoite osoittimeen.

Mietipä seuraavaa analogiaa: ystäväsi antaa sinulle numeron 800 Acme Mail Order -yhtiöön. Menet kotiin ja ohjelmoit numeron puhelimesi napin alaisuuteen; sitten voit heittää numerolapun menemään. Kun valitset numeron, puhelin soi vastapäässä ja Acme Mail Order vastaa. Et muista numeroa etkä tiedä, missä vastaajapuhelin sijaitsee, mutta puhelimen nappula antaa sinulle pääsyn Acme Mail Order -yhtiöön. Nyt Acme Mail Order on vapaassa muistissa oleva tietosi. Et tiedä, missä se on, mutta tiedät, kuinka sinne päästään. Käytät siis sen osoitetta - tässä tapauksessa puhelinnumeroa. Sinun ei tarvitse tietää numeroa, vaan sinun on pelkästään laitettava se osoittimeen - eli puhelimen napin taakse. Osoitin antaa sinulle pääsyn tietoosi eikä sinun tarvitse tietää yksityiskohtia.

Pino tyhjennetään automaattisesti, kun funktio päättyy. Kaikkien paikallisten muuttujien elinikä päättyy ja ne poistetaan pinosta. Vapaata tilaa ei tyhjennetä ennen ohjelman päättymistä, joten on sinun vastuullasi vapauttaa varattu muistitila, kun sitä ei enää tarvita.

Vapaan tilan etuna on se, että muisti on käytettävissä sen ulkoiseen vapauttamiseen saakka. Vaikka varaisit tilaa funktion sisällä, se on käytettävissä vielä funktion päättymisen jälkeenkin.

Etuna muistin käytölle tällä tavoin globaalien muuttujien sijaan on se, että funktioilla, joilla on pääsy osoittimeen, on myös pääsy tuohon tietoon. Näin saadaan hyvin tiukasti kontrolloitu liittymä tuohon tietoon. Samalla ehkäistään ongelma, jossa jokin funktio pääsee odottamattomalla tavalla käsiksi tuohon tietoon.

Jotta tuo kaikki toimisi, on voitava luoda vapaan muistitilan osoitin ja vietävä se funktioille. Seuraavat jaksot kuvaavat, kuinka se tehdään.

Komento new

Vapaasta tilasta varataan muistia varatulla sanalla new, jonka jälkeen kerrotaan varattavan kohteen tyyppi, jolloin kääntäjä saa tietää varattavan alueen koon. Siksi esimerkiksi lause new unsigned short int varaa 2 tavua muistitilaa ja lause new long taas 4 tavua.

new palauttaa muistiosoitteen, joka on sijoitettava osoittimeen. Voit luoda muistitilan tyypille unsigned short int seuraavasti:

```
unsigned short int *pPointer;
pPointer = new unsigned short int;
```

Voit tietenkin alustaa osoittimen heti esittelyn yhteydessä:

```
unsigned short int *pPointer = new unsigned short int;
```

Nyt pPointer osoittaa unsigned short int -tyyppiselle tiedolle varatun muistialueen osoitteeseen. Voit käyttää osoitinta kuten mitä tahansa johonkin muuttujaan osoittavaa osoitinta ja sijoittaa alueelle arvon seuraavasti:

```
*pPointer = 72;
```

Koodi tarkoittaa "sijoita 72 osoittimessa pPointer olevaan arvoon" tai "sijoita 72 vapaan muistin alueelle, johon pPointer osoittaa".

Jos new ei voi varata muistia - muisti on rajallinen resurssi sekin - se palauttaa null-osoittimen. Sinun on tarkistettava, osoittaako osoitin null-arvoon aina uutta muistia varatessasi.

Varoitus! Joka kerta, kun varaat muistia new-komennolla, tarkista, ettei osoitin ole null-osoitin.

Komento delete

Kun et enää tarvitse varaamaasi muistia, sinun on vapautettava muisti delete-komennolla. Muista, että osoitin itse on paikallinen muuttuja. Kun funktio, jossa osoitin on esitelty, päättyy, myös osoitin kadotetaan. new-operaattorilla varattua muistia ei vapauteta automaattisesti, vaan se jää käyttökelvottomaksi eli syntyy muistivuoto-niminen tilanne. Nimitys johtuu siitä, että muistia ei voida palauttaa käyttöön ennen ohjelman päättymistä.

Muisti palautetaan operaattorilla delete. Esimerkiksi:

```
delete pPointer;
```

Kun tuhoat osoittimen, vapautat muistin, jonka osoite on tallennettu tuohon osoittimeen. Menettelyssä "vapautetaan se muisti, johon osoitin osoittaa". Osoitin on edelleenkin osoitin ja siihen voidaan sijoittaa uusia osoitteita. Listaus 9.4 esittelee muistin varaamista keosta muuttujan avulla, muuttujan käyttöä ja sen tuhoamista.

Varoitus! Kun käytät delete-operaattoria osoittimen tuhoamiseen, osoittimen osoittava muistialue vapautetaan. Jos deleteä käytetään uudelleen tuon osoittimen suhteen, ohjelma kaatuu! Kun tuhoat osoittimen, aseta se null-osoittimeksi. Delete-operaattorin käyttäminen null-osoittimelle on varmasti turvallista. Esimerkki:

```
Animal *pDog = new Animal;
delete pDog;  // vapauttaa muistin
pDog = 0; // asetetaan osoitin osoittamaan null-arvoon
…
delete pDog;  // vaaratonta
```

Listaus 9.4. Osoittimen varaaminen ja tuhoaminen.

```
1:  // Listaus 9.4
2:  // Osoittimen allokointi ja tuhoaminen
3:
4:  #include <iostream.h>
5:  int main()
6:  {
7:    int localVariable = 5;
8:    int * pLocal= &localVariable;
9:    int * pHeap = new int;
10:   if (pHeap == NULL)
11:   {
12:     cout << "Error! No memory for pHeap!!";
13:     return 1;
14:   }
15:   *pHeap = 7;
16:   cout << "localVariable: " << localVariable << "\n";
17:   cout << "*pLocal: " << *pLocal << "\n";
18:   cout << "*pHeap: " << *pHeap << "\n";
19:   delete pHeap;
20:   pHeap = new int;
21:   if (pHeap == NULL)
22:   {
23:     cout << "Error! No memory for pHeap!!";
24:     return 1;
25:   }
26:   *pHeap = 9;
27:   cout << "*pHeap: " << *pHeap << "\n";
28:   delete pHeap;
29:   return 0;
30: }
```

Tulostus

```
localVariable: 5
*pLocal: 5
*pHeap: 7
*pHeap: 9
```

Analyysi

Rivillä 7 esitellään ja alustetaan paikallinen muuttuja. Rivi 8 esittelee ja alustaa osoittimen paikallisen muuttujan osoitteella. Rivi 9 esittelee toisen osoittimen, joka alustetaan new int -lauseella saadulla osoitteella. Tällöin varataan muistia keosta int-tyypille. Rivi 10 todentaa, että muistia on varattu ja osoitin on sopiva (ei null). Jos muistia ei ole varattu, osoitin on null-osoitin ja tulostetaan virheilmoitus.

Yksinkertaisuuden takia ei virheen tarkistusta useinkaan tehdä tulevissa ohjelmissa, mutta jonkinlainen virheen tarkistus tulee sinun sisällyttää omiin ohjelmiisi.

Rivi 15 sijoittaa arvon 7 juuri varattuun muistiin. Rivi 16 tulostaa paikallisen muuttujan arvon ja rivi 17 arvon, johon pLocal osoittaa. Arvot ovat samat, kuten odotimmekin. Rivi 19 tulostaa pHeap-osoittimen osoittaman arvon. Lause osoittaa, että rivillä 15 sijoitettu arvo on todellakin käsiteltävissä.

Rivillä 19 vapautetaan rivillä 9 varattu muisti kutsumalla delete-operaattoria. Nyt osoitin ei enää osoita tuolle muistialueelle, vaan sen voi laittaa osoittamaan jonnekin muualle, kuten riveillä 20 ja 26 tehdään. Rivi 27 tulostaa tulokset. Rivi 28 vapauttaa muistin.

Vaikka rivi 28 onkin toistoa (ohjelman päättyminen olisi vapauttanut tuon muistin), on hyvä vapauttaa muisti ulkoisesti, erillisellä lauseella. Jos ohjelma muuttuu tai sitä laajennetaan, on positiivista, että tuosta vaiheesta oli jo huolehdittu.

Muistivuodot

Toinen mahdollisuus, jossa voidaan vahingossa aikaansaada muistivuoto, on tilanne, jossa osoitin laitetaan osoittamaan muualle ennen sen muistin vapauttamista, johon osoitin aiemmin osoittaa. Katso seuraavaa esimerkkiä:

```
1: unsigned short int *pPointer = new unsigned short int;
2: *pPointer = 72;
3: delete ppOinter;
4. pPointer = new unsigned short int;
5: pPointer = 84;
```

Nyt pPointer-osoittimen osoittama muisti tuhotaan - ja vapautetaan - rivillä 4.

Huom! Aina, kun kutsut new-operaattoria, tulisi sitä vastata delete-operaattori jossain kohtaa ohjelmaa. On tärkeää seurata, mikä osoitin omistaa muistin ja varmistaa, että muisti myös vapautetaan sen jälkeen, kun sitä ei enää tarvita.

Yhteenveto

Osoittimet tarjoavat tehokkaan keinon käsitellä tietoa epäsuorasti. Jokaisella muuttujalla on osoite, joka saadaan esille osoiteoperaattorilla (&). Osoite voidaan tallentaa osoittimeen.

Osoittimet esitellään kirjoittamalla osoitettava tietotyyppi (tai kohde), asteriski (*) sekä osoittimen nimi. Osoittimet tulisi alustaa osoittamaan johonkin kohteeseen tai NULL-arvoon (0).

Osoittimen tallentamassa osoitteessa olevaa arvoa voidaan käsitellä epäsuoruusoperaattoria (*) käyttäen. Voit esitellä const-osoittimia, jotka voidaan laittaa osoittamaan uudelleen muihin kohteisiin sekä osoittimia const-kohteisiin, joita ei voida käyttää muuttamaan osoitettuja kohteita.

Muistia voidaan varata keosta osoittimien avulla new-operaattorilla. Osoittimeen sijoitetaan tällöin varatun muistialueen osoite. Tuo muisti vapautetaan delete-operaattorilla. Tällöin osoitinta ei tuhota. Siksi osoitin on laitettava osoittamaan toiseen kohteeseen muistin vapauttamisen jälkeen.

Kysymyksiä ja Vastauksia

K
Miksi osoittimet ovat niin tärkeitä?
V
Tämän luvun myötä sait nähdä, että osoittimia käytetään osoitteiden tallentamiseen. Niitä käytetään myös argumenttien viemiseen funktioille viittauksina. Luvussa 13, "Kehittyneet funktiot", kerrotaan, kuinka osoittimia käytetään luokkien monimuotoisuudessa.

K
Miksi minun tulisi käyttää vapaata muistialuetta?
V
Vapaalla muistialueella olevat kohteet säilyvät funktion päättymisen jälkeen. Lisäksi mahdollisuus tallentaa kohteita vapaaseen muistiin sallii lisäkohteiden luomisen ajon aikana eikä kohteiden määrää tällöin tarvitse tietää etukäteen. Tätä seikkaa tutkitaan tarkemmin luvussa 10.

K
Miksi jokin kohde tulisi esitellä const-tyyppisenä, jos se kerran rajoittaa tekemisiäni?
V
Ohjelmoinnissa on kyettävä hyödyntämään kääntäjää virheiden haussa. Eräs vaikeasti havaittava virhe syntyy, kun jokin funktio muuttaa kohdetta tavalla, joka ei ole kutsuvan funktion tiedossa. Kun kohde esitellään const-tyyppisenä, päästään irti sellaisista muutoksista.