Osa
IV

15. oppitunti

Taulukot

Aiemmissa luvuissa on käytetty yksinkertaisia int-, char-, yms. kohteita. Usein on kuitenkin tarvetta käyttää kohteiden kokoelmia, kuten joukkoa, jossa on 20 int-arvoa tai tusina CAT-oliota. Tässä luvussa käsitellään seuraavia aiheita:

- ❏ Mitä taulukot ovat ja kuinka niitä esitellään
- ❏ Mitä merkkijonot ovat ja kuinka merkkitaulukoita käytetään niiden luomiseen
- ❏ Taulukoiden ja osoittimien väliset suhteet
- ❏ Kuinka osoitinaritmetiikkaa käytetään taulukoiden yhteydessä

Mikä on taulukko?

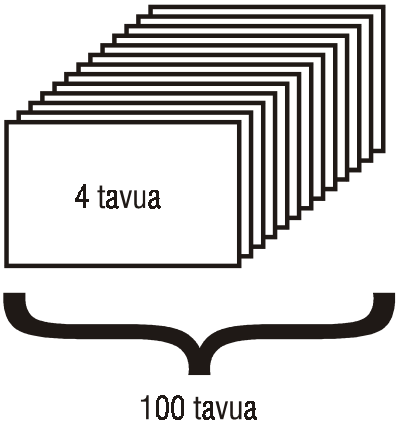
Uusi käsite Taulukko on kokoelma tiedon talletuspaikkoja, joista jokaiseen voidaan tallentaa samaa tietotyyppiä oleva tieto. Jokaista talletuspaikkaa kutsutaan taulukon alkioksi.

Uusi käsite Taulukko esitellään antamalla tietotyyppi ja sen jälkeen taulukon nimi ja lopuksi taulukon indeksit (koko, mitat, ulottuvuudet, dimensiot). Taulukon indeksi kertoo alkioiden määrän. Indeksiarvot sijoitetaan hakasulkujen ([ ]) sisään.

Esimerkiksi

```
long LongArray[25];
```

esittelee taulukon nimeltä LongArray, johon voidaan tallentaa 25 long-arvoa. Kun kääntäjä kohtaa tämän esittelyn, se varaa muistia 25 long-alkiolle. Koska jokainen long-tyyppi vie tilaa 4 tavua, varataan muistia 100 peräkkäistä tavua, kuten kuva 15.1 esittää.



Kuva 15.1. Taulukon esittely.

Taulukon alkiot

Kutakin taulukon alkiota voidaan käsitellä viittaamalla siirtymään taulukon sisällä. Taulukon alkiot lasketaan alkaen indeksistä 0, joten ensimmäisen alkion paikka on TaulukonNimi[0]. Käytettäessä LongArray-esimerkkiä viitataan ensimmäiseen alkioon siis merkinnällä LongArray[0], toinen alkio on kohdassa LongArray[1], jne.

Viittaaminen voi tuntua aluksi hämäävältä. Taulukossa SomeArray[3] on kolme alkiota: SomeArray[0], SomeArray[1] ja SomeArray[2]. Yleisesti sanottuna on taulukossa SomeArray[n] n alkiota, jotka ovat väliltä SomeArray[0] ja SomeArray[n-1].

Siksi LongArray[25]-taulukon alkiot ovat väliltä LongArray[0] ja LongArray[24]. Listaus 15.1 esittelee 5 int-alkiota sisältävän taulukon ja sijoittaa arvot taulukkoon.

Listaus 15.1. Kokonaislukutaulukon käyttö.

```
1:  //Listaus 15.1 - Taulukot
2:  #include <iostream.h>
3:
4:  int main()
5:  {
6:    int myArray[5];
7:    for (int i=0; i<5; i++)  // 0-4
8:    {
9:      cout << "Value for myArray[" << i << "]: ";
10:     cin >> myArray[i];
11:   }
12:   for (i = 0; i<5; i++)
13:     cout << i << ": " << myArray[i] << "\n";
14:   return 0;
15: }
```

Tulostus

```
Value of myArray[0]:  3
Value of myArray[1]:  6
Value of myArray[2]:  9
Value of myArray[3]:  12
Value of myArray[4]:  15

0: 3
1: 6
2: 9
3: 12
4: 15
```

Analyysi

Rivillä 6 esitellään taulukko nimeltä myArray, johon voidaan tallentaa viisi kokonaislukua. Rivillä 7 on silmukka, joka laskee nollasta neljään eli käy sopivasti läpi taulukon indeksit. Käyttäjää pyydetään antamaan arvoja, jotka tallennetaan oikeaan paikkaan taulukossa.

Ensimmäinen arvo tallennetaan kohtaan myArray[0], toinen kohtaan myArray[1], jne. Toinen for-silmukka tulostaa sitten taulukon arvot.

Huom! Taulukon indeksi alkaa arvosta 0 eikä 1. Tämä aiheuttaa usein virheitä aloittelevalle ohjelmoijalle. Muista, että taulukon TaulukonNimi[10] alkiot ovat väliltä TaulukonNimi[0] ja TaulukonNimi[9]. Sellaista alkiota kuin TaulukonNimi[10] ei ole nyt olemassa.

Viittaaminen taulukon rajojen ulkopuolelle

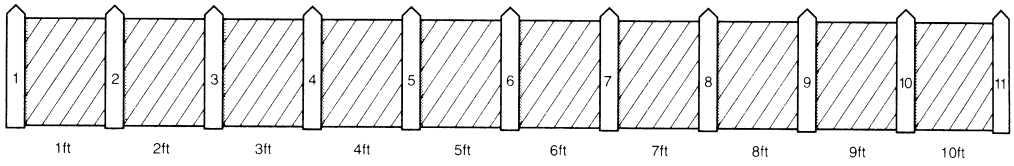
Kun taulukon alkioon sijoitetaan arvo, kääntäjä laskee paikan taulukon koon ja indeksin mukaan. Olettakaamme, että jokin long-arvo sijoitetaan kohtaan LongArray[5] eli siis 6. alkioon. Kääntäjä kertoo siirtymän - eli arvon 5 - alkion koolla - joka on nyt siis 4 tavua (long-tyypin koko). Sitten

kääntäjä siirtyy nuo 20 tavua taulukon alusta ja kirjoittaa uuden arvon tuohon paikkaan.

Jos koodissa pyydetään tallentamaan arvo kohtaan LongArray[50], kääntäjä ei välitä siitä tosiseikasta, että sellaista alkiota ei ole olemassakaan. Se vain laskee paikan edellä kuvatulla tavalla (nyt 200 tavua eteenpäin) ja tekee tallennuksen tuohon paikkaan muistissa. Arvon kirjoittaminen tuollaiseen paikkaan voi aiheuttaa ennalta aavistamattomia tuloksia. Jos käy hyvin, ohjelma kaatuu välittömästi. Jos taas on epäonnea, ohjelma alkaa antaa outoja tuloksia joskus myöhemmin ja tällöin voi virheiden etsiminen olla työläämpää.

Kääntäjä on tässä suhteessa kuin sokea mies. Jos miehelle annetaan ohjeet siirtyä viidenteen kortteliin tietyllä kadulla, mies laskee askeleiden määrästä kyseisen korttelin paikan alkaen ensimmäisestä korttelista, jonka indeksi on nolla. Jos kortteleita on kuitenkin vain 3 kappaletta kyseisellä kadulla, mies voi joutua rantaan, jossa odottaa Suomenlinnan vesibussi. Ole siis tarkka indeksien suhteen.

Juuri viimeistä alkiota seuraavaan kohtaan kirjoittaminen on niin yleinen virhe, että sille on annettu oma nimensä.



Kuva 15.2 kuvaa tätä ongelmaa.

On aina muistettava, että 25-alkioisen taulukon alkioiden indeksit ovat väliltä 0 - 24 ja laskeminen siis alkaa indeksistä 0. (Monet ohjelmoijat ihmettelevät usein, miksei rakennuksissa ole kerrosta nolla. Joidenkin on nähty painavan hissin nappia numero 4, kun he ovat halunneet mennä 5. kerrokseen.)

Taulukoiden alustaminen

Taulukko voidaan alustaa yksinkertaista tyyppiä olevilla arvoilla esittelyn yhteydessä. Taulukon nimen jälkeen annetaan yhtäsuuruusmerkki ja sen jälkeen aaltosulkujen sisälle luettelo arvoista pilkuilla erotettuina. Esimerkiksi:

```
int IntegerArrays[5] = { 10, 20, 30, 40, 50 };
```

esittelee taulukon, joka sisältää 5 kokonaislukua. Alkioon IntegerArray[0] sijoitetaan arvo 10, alkioon IntegerArray[1] arvo 20, jne.

Jos esittelyssä ei anneta taulukon kokoa, muodostetaan taulukko, johon mahtuvat alustusarvot. Jos siis kirjoitat esimerkiksi

```
int IntegerArray[] = { 10, 20, 30, 40, 50 };
```

luodaan samankokoinen taulukko kuin edellisessä esimerkissä.

Jos taulukon koko halutaan tietää, voidaan kääntäjältä kysyä sitä. Esimerkiksi

```
const USHORT IntegerArrayLength=sizeof (IntegerArray)/sizeof
(IntegerArray[0]);
```

asettaa vakioon USHORT-muuttujaan IntegerArrayLength taulukon koon, joka saadaan jakamalla koko taulukon viemä muistitila yhden alkion viemällä muistitilalla. Tuloksena on alkioiden määrä.

Taulukossa ei voida alustaa enempää alkioita kuin esittely sallii. Siksi

```
int IntegerArray[5] = { 10, 20, 30, 40, 50, 60 };
```

aiheuttaa kääntäjän virheen, koska alustusarvoja on 6 esittelyn salliessa 5 alkiota. Sen sijaan on täysin sallittua kirjoittaa

```
int IntegerArray[5] = { 10, 20 };
```

Vaikka alustamattomalla taulukolla ei olekaan arvoja, sijoitetaan jäseniin kuitenkin arvot 0.

Oliotaulukot

Taulukkoon voidaan tallentaa mitä tahansa kohteita, joko sisäisiä tai käyttäjän määrittämiä. Kun taulukko esitellään, kerrotaan kääntäjälle tallennettavan kohteen tyyppi sekä määrä, ja kääntäjä varaa vastaavan määrän muistitilaa. Olioiden kohdalla kääntäjä tietää varattavan tilan luokan esittelystä. Luokalla täytyy olla oletusmuodostin, joka ei ota argumentteja, jotta olioita voidaan luoda taulukkoa määritettäessä.

Oliotaulukossa olevia jäsenmuuttujia käsitellään kaksivaiheisessa prosessissa. Taulukon jäsen identifioidaan indeksioperaattorilla ([ ]), jonka jälkeen tulee jäsenoperaattori (.), jolla käsitellään tiettyä jäsenmuuttujaa. Listaus 15.2 esittelee, kuinka esitellään viisi CAT-oliota sisältävä taulukko.

Listaus 15.2. Oliotaulukon luominen.

```
1:  // Listaus 15.2 - Oliotaulukko
2:
3:  #include <iostream.h>
4:
5:  class CAT
6:  {
7:  public:
8:    CAT() { itsAge = 1; itsWeight=5; }        // oletusmuodostin
9:    ~CAT() {}                                 // tuhoaja
10:   int GetAge() const { return itsAge; }
11:   int GetWeight() const { return itsWeight; }
12:   void SetAge(int age) { itsAge = age; }
13:
14: private:
15:   int itsAge;
16:   int itsWeight;
17: };
18:
19: int main()
20: {
21:   CAT Litter[5];
22:   int i;
23:   for (i = 0; i < 5; i++)
24:     Litter[i].SetAge(2*i +1);
25:
26:   for (i = 0; i < 5; i++)
27:     cout << "Cat #" << i+1<< ": " << Litter[i].GetAge() << endl;
28:   return 0;
29: }
```

Tulostus

```
cat #1:  1
cat #2:  2
cat #3:  3
cat #4:  4
cat #5:  5
```

Analyysi

CAT-luokka esitellään riveillä 5-17. Luokalla tulee olla oletusmuodostin, jotta CAT-olioita voidaan luoda taulukkoon. Muista, että, jos luot toisen muodostimen, kääntäjän tarjoamaa oletusmuodostinta ei luoda; sinun on tällöin luotava oma oletusmuodostin.

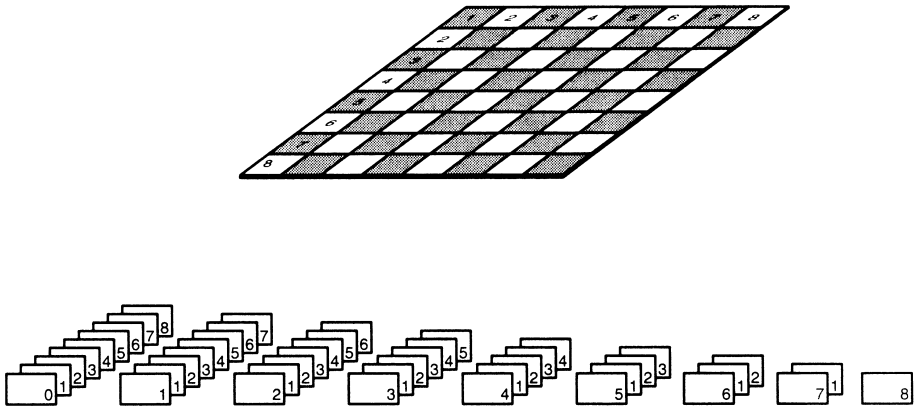
Ensimmäisessä silmukassa (rivit 23-24) asetetaan kunkin taulukossa olevan CAT-olion ikä. Toinen for-silmukka (rivit 26-27) käsittelee kutakin taulukon jäsentä ja kutsuu jäsenfunktiota GetAge().

Kunkin yksittäisen CAT-olion GetAge()-metodia kutsutaan käsittelemällä taulukon vastaavaa jäsentä, Litter[i], jota seuraa pisteoperaattori (.) ja jäsenfunktio.

Moniulotteiset taulukot

Taulukossa voi olla enemmänkin kuin yksi ulottuvuus. Kukin ulottuvuus esitetään taulukon indeksinä. Siksi kaksiulotteisella taulukolla on kaksi indeksiä, kolmeulotteisella kolme indeksiä, jne. Taulukolla voi olla lukematon määrä ulottuvuuksia, mutta todennäköisesti enintään kolme kappaletta.

Hyvä esimerkki kaksiulotteisesta taulukosta on shakkipöytä. Yksi ulottuvuus esittää 8 riviä ja toinen taas 8 saraketta. Kuva 15.3 havainnollistaa tätä.



Kuva 15.3. Shakkipöytä ja kaksiulotteinen taulukko.

Olettakaamme, että meillä on luokka nimeltä SQUARE. Taulukon nimeltä Board esittely voisi olla seuraavanlainen:

```
SQUARE Board[8][8];
```

Sama tieto voitaisiin esittää myös yksiulotteisena taulukkona, jossa on 64 alkiota. Esimerkiksi:

```
SQUARE Board[64];
```

Tämä ei vastaa täysin käytännön kohdetta, kuten kaksiulotteinen taulukko. Kun peli alkaa, sijaitsee kuningas ensimmäisen rivin neljännessä paikassa. Kun laskeminen alkaa nollasta, olisi paikka siis

```
Board[0][3];
```

Olettaen, että ensimmäinen indeksi kuvaa rivejä ja toinen sarakkeita. Koko shakkipöydän kaikkia paikkoja esittää kuva 15.3.

Moniulotteisen taulukon alustaminen

Myös moniulotteiset taulukot voidaan alustaa. Arvot sijoitetaan alkioihin oikeassa järjestyksessä, jolloin viimeinen indeksi muuttuu aiempien indeksien pysyessä muuttumattomina. Jos meillä siis on taulukko

```
int theArray[5][3]
```

menevät kolme ensimmäistä alkiota kohtaan theArray[0], seuraavat kolme kohtaan theArray[1], jne.

Taulukko alustetaan kirjoittamalla

```
int theArray[5][3] = { 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15 }
```

Selvyyden vuoksi voisit ryhmitellä alustukset aaltosuluilla. Esimerkiksi:

```
int theArray[5][3] = { { 1,2,3},
{4,5,6},
{7,8,9},
{10,11,12},
{13,14,15 } };
```

Kääntäjä ohittaa sisäiset aaltosulut, joiden tarkoituksena on vain selkeyttää taulukon esittelyä.

Kukin arvo tulee erottaa pilkulla huolimatta aaltosuluista. Koko alustus tulee sijoittaa aaltosulkuihin ja päättää puolipisteellä.

Listaus 15.3 luo kaksiulotteisen taulukon. Ensimmäinen ulottuvuus sisältää arvot 0-4 ja toinen samat arvot tuplana.

Listaus 15.3. Moniulotteisen taulukon luominen.

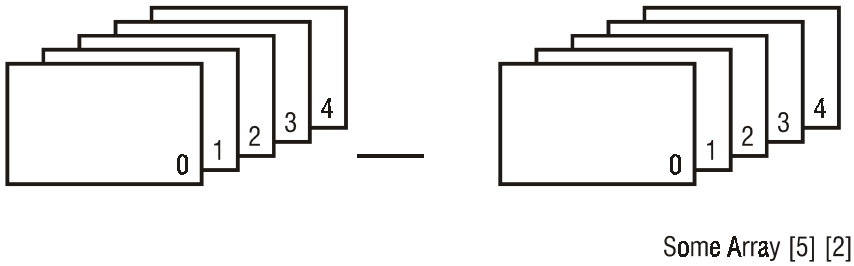
```
1:  #include <iostream.h>
2:  int main()
3:  {
4:    int SomeArray[5][2] = { {0,0}, {1,2}, {2,4}, {3,6}, {4,8}};
5:    for (int i = 0; i<5; i++)
6:      for (int j=0; j<2; j++)
7:      {
8:        cout << "SomeArray[" << i << "][" << j << "]: ";
9:        cout << SomeArray[i][j]<< endl;
10:     }
11:
12:   return 0;
13: }
```

Tulostus

```
SomeArray[0][0]: 0
SomeArray[0][1]: 0
SomeArray[1][0]: 1
SomeArray[1][1]: 2
```

```
SomeArray[2][0]: 2
SomeArray[2][1]: 4
SomeArray[3][0]: 3
SomeArray[3][1]: 6
SomeArray[4][0]: 4
SomeArray[4][1]: 8
```

Analyysi
Rivi 4 esittelee taulukon SomeArray kaksiulotteisena taulukkona. Ensimmäinen indeksi sisältää 5 kokonaislukua ja toinen kaksi kokonaislukua. Tällöin syntyy 5x2-ruudukko. Kuva 15.4 havainnollistaa taulukkoa.



Kuva 15.4. 5x2-taulukko.

Arvot alustetaan pareittain, vaikka ne olisi voitu laskeakin. Rivit 5 ja 6 luovat sisäkkäiset silmukat. Ulompi for-silmukka käy läpi ensimmäisen indeksin. Jokaisen ensimmäisen indeksin kohdalla käy sisempi for-silmukka läpi kaikki toisen indeksin jäsenet. Tulostus näyttää tämän: alkiota SomeArray[0][0] seuraa alkio SomeArray[0][1]. Ensimmäistä ulottuvuutta kasvatetaan sen jälkeen, kun toista ulottuvuutta on kasvatettu yhdellä.

Muutama sana muistista

Kun esittelet taulukon, kerrot kääntäjälle tarkkaan, kuinka monta kohdetta taulukkoon aiotaan tallentaa. Kääntäjä varaa muistia kaikille kohteille, vaikket käyttäisikään sitä koskaan. Tämä ei ole ongelma, kun tiedät etukäteen, kuinka monta alkiota tarvitset. Esimerkiksi shakkipöytää varten tarvitaan 64 paikkaa. Jos et tiedä kohteiden määrää edes arviolta, on sinun kuitenkin käytettävä kehittyneempiä tietorakenteita.

Tässä kirjassa käsitellään osoitintaulukoita, vapaaseen muistiin muodostettuja taulukoita sekä lukuisia muita kokoelmia. Muut, vielä kehittyneemmät tietorakenteet, jotka ratkaisevat suurten tietomäärien tallennusongelmia, ei tässä kirjassa käsitellä. Ohjelmoinnin kaksi hyvää asiaa ovatkin seuraavat: aina on lisää opittavaa ja aina on myös uutta kirjallisuutta tutkittavaksi.

Osoitintaulukot

Tähän mennessä käsitellyt taulukot ovat tallentaneet alkionsa pinoon. Yleensä pinomuisti on rajallista, kun taas vapaa muisti on paljon laajempi. On mahdollista esitellä kukin kohde vapaaseen muistiin ja tallentaa vain osoitin tuohon kohteeseen taulukkoon. Tämä menettely vähentää jyrkästi pinomuistin käyttötarvetta. Listaus 15.4 on listaus 15.3 muutettuna siten, että kaikki taulukon oliot tallennetaan vapaaseen muistiin. Tämä menettely mahdollistaa suuremman muistin käytön ja taulukon koko onkin nyt arvon 5 sijaan 500. Samalla on taulukon nimi Litter muutettu nimeksi Family.

Listaus 15.4. Taulukon tallentaminen vapaaseen muistiin.

```
1:  // Listaus 15.4 - Olio-osoitintaulukko
2:
3:  #include <iostream.h>
4:
5:  class CAT
6:  {
7:  public:
8:    CAT() { itsAge = 1; itsWeight=5; }        // oletusmuodostin
9:    ~CAT() {}                                 // tuhoaja
10:   int GetAge() const { return itsAge; }
11:   int GetWeight() const { return itsWeight; }
12:   void SetAge(int age) { itsAge = age; }
13:
14: private:
15:   int itsAge;
16:   int itsWeight;
17: };
18:
19: int main()
20: {
21:   CAT * Family[500];
22:   int i;
23:   CAT * pCat;
24:   for (i = 0; i < 500; i++)
25:   {
26:     pCat = new CAT;
27:     pCat->SetAge(2*i +1);
28:     Family[i] = pCat;
29:   }
30:
31:   for (i = 0; i < 500; i++)
32:     cout << "Cat #" << i+1 << ": " << Family[i]->GetAge() << endl;
33:   return 0;
34: }
```

Tulostus

```
Cat #1:  1
Cat #2:  3
Cat #3:  5
…
Cat #499:  997
Cat #500:  999
```

Analyysi
CAT-olio esitellään riveillä 5-17. Se on samanlainen kuin listauksessa 15.2. Tällä erää on rivillä 21 esitellyn taulukon nimenä Family ja se esitellään tallentamaan 500 osoitinta CAT-olioihin.

Alustussilmukassa (rivit 24-29) luodaan 500 uutta CAT-oliota vapaaseen muistiin ja kunkin olion iäksi asetetaan indeksi + yksi. Siksi ensimmäisen CAT-olion iäksi tulee 1, toisen 2, jne. Lopuksi osoitin lisätään taulukkoon.

Koska taulukko on esitelty osoitintaulukkona, lisätään osoitin - ei sen osoittamaa arvoa - taulukkoon.

Toinen silmukka (rivit 31-32) tulostaa kunkin arvoista. Osoitinta käsitellään indeksin avulla, Family[i]. Tuota osoitetta käytetään sitten GetAge()-metodia kutsuttaessa.

Tässä esimerkissä tallennetaan Family-taulukko sekä kaikki sen osoittimet pinoon, mutta 500 CAT-oliota tallennetaan vapaaseen muistiin.

Taulukoiden luonti vapaaseen muistiin

Koko taulukko on mahdollista sijoittaa vapaaseen muistiin, jota kutsutaan myös nimellä keko. Se toteutetaan kutsumalla operaattoria new. Tuloksena on osoitin vapaaseen muistitilaan, jossa taulukko on. Esimerkiksi:

```
CAT *Family = new CAT[500];
```

esittelee Familyn osoittimena 500 CAT-oliotaulukon ensimmäiseen alkioon. Toisin sanoen Family osoittaa kohteeseen Family[0] (tai sillä on kohteen Family[0] osoite).

Etuna on nyt se, että voimme hyödyntää osoitinaritmetiikkaa Family-jäsenten käsittelyssä. Voit kirjoittaa esimerkiksi:

```
CAT *Family = new CAT[500];
CAT *pCat = Family; //pCat osoittaa alkioon Family[0]
pCat->SetAge(10);   //asettaa Family[0]:n iän arvoksi 10
pCat++;     // siirtyminen osoittamaan kohteeseen Family[1]
pCat->SetAge(20);   // asettaa Family[1]:n iän arvoksi 20
```

Edellä esitellään 500 CAT-oliota sisältävä taulukko ja osoitin osoittaa taulukon alkuun. Osoittimen avulla kutsutaan ensimmäisen CAT-olion SetAge()-funktiota asettamaan iäksi 10. Sitten osoitinta kasvatetaan osoittamaan toiseen CAT-olioon ja kutsutaan sen SetAge()-metodia.

Osoitin taulukkoon vai osoittimia sisältävä taulukko

Tutki seuraavia kolmea esittelyä:

```
1: CAT FamilyOne[500];
2: CAT * FamilyTwo[500];
3: CAT * FamilyThree = new CAT[500];
```

FamilyOne on 500 CAT-oliota sisältävä taulukko. FamilyTwo on taulukko, joka sisältää 500 CAT-olioihin sisältävää osoitinta. FamilyThree on osoitin taulukkoon, jossa on 500 CAT-oliota.

Noiden koodirivien väliset erot vaikuttavat dramaattisesti siihen, kuinka taulukoita käytetään. Vielä hämmästyttävämpää saattaa olla se, että FamilyThree on FamilyOne-muunnos, mutta eroaa hyvin paljon FamilyTwo-esittelystä.

Tästä nouseekin keskustelu siitä, kuinka osoittimet ja taulukot liittyvät yhteen. FamilyThree on osoitin taulukkoon. Eli FamilyThree-osoittimen osoite on tuon taulukon ensimmäisen alkion osoite. Juuri näin on myös FamilyOne-esittelyn kohdalla.

Osoittimet ja taulukoiden nimet

C++ -kielessä on taulukon nimi osoitin taulukon ensimmäiseen alkioon. Siksi esittelyssä

```
CAT Family[50];
```

on Family osoitin &Family[0]:aan, joka on Family-taulukon ensimmäisen alkion osoite.

On laillista käyttää taulukoiden nimiä vakio-osoittimina ja kääntäen. Siksi Family + 4 on hyväksyttävä tapa käsitellä alkion Family[4] tietoa.

Kääntäjä hoitaa kaiken aritmetiikan laskettaessa yhteen, kasvatettaessa ja vähennettäessä osoittimia. Osoite, johon Family + 4 viittaa, ei ole 4 tavua alusta päin, vaan 4 oliota eteenpäin. Jos kukin olio on 4 tavun kokoinen, on Family + 4 16 tavua. Jos kukin olio on CAT, jossa on neljä long-jäsenmuuttujaa (kukin 4 tavun kokoinen) sekä kaksi short-jäsenmuuttujaa (kukin 2 tavun kokoinen), on CAT-olion koko 20 tavua ja Family + 4 on 80 tavua alusta eteenpäin.

Listaus 15.5 havainnollistaa vapaassa muistissa olevan taulukon esittelyä ja käyttöä.

Listaus 15.5. Taulukon luonti new-operaattorilla.

```
1:  // Taulukko new-komennolla
2:
3:  #include <iostream.h>
4:
5:  class CAT
6:  {
7:  public:
8:    CAT() { itsAge = 1; itsWeight=5; }      // oletusmuodostin
9:    ~CAT();                                 // tuhoaja
10:   int GetAge() const { return itsAge; }
11:   int GetWeight() const { return itsWeight; }
12:   void SetAge(int age) { itsAge = age; }
13:
14: private:
15:   int itsAge;
16:   int itsWeight;
17: };
18:
19: CAT :: ~CAT()
20: {
21:   // cout << "Destructor called!\n";
22: }
23:
24: int main()
25: {
26:   CAT * Family = new CAT[500];
27:   int i;
28:   CAT * pCat;
29:   for (i = 0; i < 500; i++)
30:   {
31:     pCat = new CAT;
32:     pCat->SetAge(2*i +1);
33:     Family[i] = *pCat;
34:     delete pCat;
35:   }
36:
37:   for (i = 0; i < 500; i++)
38:     cout << "Cat #" << i+1 << ": " << Family[i].GetAge() << endl;
39:
40:   delete [] Family;
41:
42:   return 0;
43: }
```

Tulostus

```
Cat # 1: 1
Cat # 2: 3
Cat # 3: 5

Cat # 499: 997
Cat # 500: 999
```

Analyysi

Rivi 26 esittelee Family-Taulukon, joka sisältää 500 CAT-oliota. Koko taulukko luodaan vapaaseen muistiin new CAT[500] -kutsulla.

Jokainen taulukkoon lisätty CAT-olio luodaan myös vapaaseen muistiin (rivi 31). Huomaa, että osoitinta ei tällä kertaa lisätä kuitenkaan taulukkoon, vaan itse olio. Tämä ei ole osoittimia sisältävä taulukko, vaan CAT-olioita sisältävä taulukko.

Taulukoiden tuhoaminen vapaasta muistitilasta

Family on osoitin vapaassa muistissa olevaan taulukkoon. Kun osoittimella pcat (rivi 33) osoitetaan uudelleen, tallennetaan CAT-olio itse taulukkoon (miksi ei? taulukko on vapaassa tilassa). Mutta pCat-osoitinta käytetään uudelleen seuraavalla silmukkakierroksella. Eikö nyt ole vaaraa, ettei tuohon CAT-olioon ole lainkaan osoitinta ja syntyy muistivuoto?

Tämä saattaisi olla suuri ongelma, paitsi että Familyn tuhoaminen vapauttaa kaiken varatun muistin. Kääntäjä osaa tuhota jokaisen olion taulukosta ja vapauttaa muistin.

Todetaksesi tämän muuta taulukon kokoa 500:sta kymmeneen riveillä 26, 29 ja 36. Ota sitten pois kommenttimerkit cout-lauseesta rivillä 21. Kun rivi 39 kohdataan ja taulukko tuhotaan, kutsutaan kunkin CAT-olion tuhoajafunktiota.

Kun luot kohteen kekoon new-operaattorilla, tuo kohde tuhotaan ja muisti vapautetaan delete-operaattorilla. Samalla lailla luodessasi taulukon lauseella new <luokka>[koko] tuhotaan taulukko ja muisti vapautetaan komennolla delete[]. Hakasulut kertovat kääntäjälle, että tuo taulukko tuhotaan.

Jos jätät hakasulut pois, vain taulukon ensimmäinen olio tuhotaan. Voit todeta tämän itse jättämällä hakasulut pois riviltä 39. Jos muutit riviä 21 siten, että tuhoajafunktion toiminta tulostuu, näet nyt, että vain yksi CAT-olio tuhotaan. Onnittelut! Loit juuri muistivuodon.

Tee/Älä tee

Muista, että n-alkioisen taulukon alkiot ovat väliltä 0 - n-1.
Älä kirjoita taulukon rajojen ulkopuolelle. Älä myöskään lue sieltä.
Älä sekoita osoitintaulukkoa taulukko-osoittimeen.
Käytä taulukon indeksointia osoittimilla, jotka osoittavat taulukoihin.

char-taulukot

Merkkijono on sarja merkkejä. Toistaiseksi olet nähnyt vain nimettömiä merkkijonovakioita, joita on käytetty cout-lauseissa, kuten

```
cout << "Hello world.\n";
```

C++ -kielessä merkkijono on merkkitaulukko, joka päättyy null-merkkiin. Voit esitellä ja alustaa merkkijonon aivan kuin tavallisen taulukon. Esimerkiksi:

```
char Greeting[] = { 'H', 'e', 'l', 'l', 'o', ' ', 'W', 'o',
'r', 'l', 'd', '\0' };
```

Viimeinen merkki, '\0', on null-merkki, jonka monet C++ -funktiot tunnistavat merkkijonon loppumerkiksi. Vaikka tämä merkkikohtainen menettely toimiikin, on sen kirjoittaminen työlästä ja virhealtista. C++ mahdollistaakin lyhyemmän kirjoitusmuodon:

char Greeting[] = "Hello World";

Huomaa kaksi seikkaa tuossa lauseessa:
Heittomerkkien, pilkkujen ja aaltosulkujen sijaan kirjoitetaan nyt koko merkkijono lainausmerkkien sisälle.
Null-merkkiä ei nyt tarvitse lisätä, koska kääntäjä lisää sen.

Merkkijono "Hello world" on 12 tavun mittainen: Hello vie 5 tavua, välilyönti 1 tavun, World 5 tavua ja null-merkki 1 tavun.

Voit luoda myös alustamattomia merkkijonotaulukoita. Kuten tavallisten taulukoiden kohdalla, on nytkin muistettava, ettei puskuriin sijoiteta enempää kuin siihen mahtuu.

Listaus 15.6 esittelee alustamattoman puskurin käyttöä.

Listaus 15.6. Taulukon täyttäminen.

```
1:  //Listaus 15.6 char-taulukkopuskuri
2:
3:  #include <iostream.h>
4:
5:  int main()
6:  {
7:    char buffer[80];
8:    cout << "Enter the string: ";
9:    cin >> buffer;
10:   cout << "Here's the buffer:  " << buffer << endl;
11:   return 0;
12: }
```

Tulostus

```
Enter the string: Hello World
Here's the buffer: Hello
```

Analyysi

Rivillä 7 esitellään buffer tallentamaan 80 merkkiä. Siihen sopii 79 merkkiä ja null-merkki.

Rivillä 8 pyydetään käyttäjää antamaan merkkijono, joka sijoitetaan puskuriin rivillä 9. cin-olio kirjoittaa lopettavan null-merkin puskuriin sen jälkeen, kun se on kirjoittanut merkkijonon.

Tässä listauksessa on pari ongelmaa. Ensiksikin, jos käyttäjä antaa enemmän kuin 79 merkkiä, cin kirjoittaa puskurin ulkopuolelle. Toiseksi, jos käyttäjä antaa välilyönnin, cin ajattelee, että kyseessä on merkkijonon loppuminen ja lopettaa kirjoittamisen puskuriin.

Nämä ongelmat voidaan ratkaista kutsumalla erikoismetodia cin:get(), joka ottaa kolme parametria:

- Täytettävän puskurin
- Maksimimerkkimäärän
- Erottimen, joka päättää syötön

Oletuserottimena on rivinvaihto. Listaus 15.7 havainnollistaa sen käyttöä.

Listaus 15.7. Taulukon täyttäminen.

```
1:  //Listaus 15.7 cin.get()
2:
3:  #include <iostream.h>
4:
5:  int main()
6:  {
7:    char buffer[80];
8:    cout << "Enter the string: ";
9:    cin.get(buffer, 79);       // get up to 79 or newline
10:   cout << "Here's the buffer:  " << buffer << endl;
11:   return 0;
12: }
```

Tulostus

```
Enter the string: Hello World
Here's the buffer: Hello World
```

Analyysi

Rivi 9 kutsuu cin-metodia get(). Ensimmäisenä parametrina viedään rivillä 7 esitelty puskuri. Toisena parametrina on saatavien merkkien maksimimäärä. Tässä tapauksessa se on 79, jolloin jää tilaa vielä lopettavalle null-merkille. Ei ole tarvetta antaa loppumerkkiä, koska oletuksena oleva rivinvaihto riittää.

cin-oliota ja sen muunnelmia käsitellään luvussa 21, "Esikäsittelijä", jossa tutkitaan virtoja tarkemmin.

strcpy() ja strncpy()

C++ perii C-kirjastofunktiot, jotka käsittelevät merkkijonoja. Kaksi noista funktioista on tarkoitettu merkkijonojen kopiointiin: strcpy() ja strncpy(). strcpy() kopioi yhden merkkijonon koko sisällön annettuun puskuriin. Listaus 15.8 esittelee sen käyttöä.

Listaus 15.8. strcpy()-funktion käyttö.

```
1:  #include <iostream.h>
2:  #include <string.h>
3:  int main()
4:  {
5:    const int MaxLength = 80;
6:    char String1[] = "No man is an island";
7:    char String2[MaxLength+1];
8:
9:    strcpy(String2,String1);
10:
11:   cout << "String1: " << String1 << endl;
12:   cout << "String2: " << String2 << endl;
13:   return 0;
14: }
```

Tulostus

```
String1: No man is an island
String2: No man is an island
```

Analyysi

Rivillä 2 sisällytetään koodiin otsikkotiedosto STRING.H, joka sisältää funktion strcpy() prototyypin. Funktio ottaa kaksi merkkijonotaulukkoa - kohde- ja lähdetaulukon. Jos lähde on suurempi kuin kohde, strcpy() kirjoittaa puskurin ulkopuolelle.

Jotta tältä suojaudutaan, standardikirjasto sisältää myös funktion strncpy(). Tämä muunnos kopioi maksimimäärän merkkejä. Se kopioi joko lopettavaan merkkiin saakka tai sitten kohdepuskuriin mahtuvan määrän.

Listaus 15.9 havainnollistaa strncpy()-funktion käyttöä.

Listaus 15.9. strncpy()-funktion käyttö.

```
1:  #include <iostream.h>
2:  #include <string.h>
3:  int main()
4:  {
5:    const int MaxLength = 80;
6:    char String1[] = "No man is an island";
7:    char String2[MaxLength+1];
8:
9:
10:   strncpy(String2,String1,MaxLength);
11:
12:   cout << "String1: " << String1 << endl;
13:   cout << "String2: " << String2 << endl;
14:   return 0;
```

```
15: }
```

Tulostus

```
String1: No man is an island
String2: No man is an island
```

Analyysi

Rivillä 10 kutsutaan nyt strncpy()-funktiota strcpy()-funktion sijaan. Funktio ottaa kolmannen parametrin, kopioitavien merkkien maksimimäärän. Puskuri String2 esitellään tallentamaan MaxLength+1 merkkiä. Ylimääräinen merkki on null, jonka molemmat funktiot laittavat automaattisesti merkkijonon loppuun.

Merkkijonoluokat

Useimmat C++ -kääntäjät sisältävät luokkakirjaston, jossa on suuri joukko luokkia tiedon käsittelyyn. Luokkakirjaston standardikomponentti on String-luokka.

C++ peri null-päätteisen merkkijonon ja funktiokirjaston, joka sisältää strcpy()-funktion, C-kielestä. Nuo funktiot on kuitenkin yhdistetty oliopohjaiseen kehykseen. String-luokka tarjoaa kapseloidun tietojoukon ja funktiot tuon tiedon käsittelyyn sekä käsittelyfunktiot, joten itse tieto on piilotettu String-luokan asiakkailta.

Jos kääntäjässäsi ei jo ole String-luokkaa (ja vaikka sillä olisikin), saatat haluta kirjoittaa omasi.

Minimimuodossaan String-luokan tulisi ylittää merkkitaulukoiden perusrajoitteet. Kuten kaikki taulukot, myös merkkitaulukot ovat staattisia. Niiden koko määritellään itse. Ne vievät aina runsaasti tilaa muistista, vaikket tarvitsisikaan kaikkea. Taulukon ulkopuolelle kirjoittaminen on vaarallista.

Hyvä String-luokka varaa tilaa vain tarvittavan määrän. Jos tilaa ei voida varata, tilanne hoidetaan kunnolla.

Yhteenveto

Tässä luvussa opit luomaan taulukoita. Taulukko on kiinteän kokoinen tietorakenne, johon tallennetaan kohteita, jotka ovat kaikki samaa tyyppiä.

Taulukoiden rajoja ei tarkisteta. Siksi onkin mahdollista -joskin vaarallista- kirjoittaa tai lukea alueelta, joka ei kuulu taulukkoon. Taulukoiden

ensimmäinen indeksi on 0. On yleinen virhe kirjoittaa indeksiksi n, kun taulukossa on n alkiota.

Taulukot voivat olla yksi- tai moniulotteisia. Molemmissa tapauksissa voidaan taulukot alustaa. Jos taulukot sisältävät olioita, tulee luokassa olla oletusmuodostin.

Taulukot ja niiden sisällöt voidaan tallentaa joko vapaaseen muistiin tai pinoon. Jos taulukko tuhotaan vapaasta muistista delete-operaattorilla, on delete-operaattorin yhteydessä muistettava käyttää hakasulkuja.

Taulukoiden nimet ovat taulukon ensimmäiseen alkioon osoittavia vakio-osoittimia. Osoittimet ja taulukot käyttävät osoitinaritmetiikkaa alkioiden viittauksissa.

Merkkijonot ovat merkkitaulukoita. C++ sisältää erikoispiirteitä merkkitaulukoiden käsittelyyn, kuten niiden alustamisen lainausmerkein varustetuilla merkkijonoilla.

Kysymyksiä ja Vastauksia

K
Mitä tapahtuu jos tallennan 25 jäsentä 24-alkioiseen taulukkoon?
V
Kirjoitat muistialueelle taulukon ulkopuolelle, mikä voi aiheuttaa vaaratilanteita ohjelmassasi.

K
Mikä on alustamaton taulukon alkio?
V
Se, mitä on muistissa kunakin ajankohtana. Tällaisen alustamattoman alkion, johon ei siis ole sijoitettu tiettyä arvoa, käyttäminen on tietenkin vaarallista.

K
Voinko yhdistää taulukoita?
V
Kyllä. Yksinkertaisten taulukoiden kohdalla voin käyttää osoittimia yhdistämään niitä uuteen, suurempaan taulukkoon. Merkkijonojen kohdalla voin käyttää sisäisiä funktioita, kuten strcat, merkkijonojen yhdistämiseen.