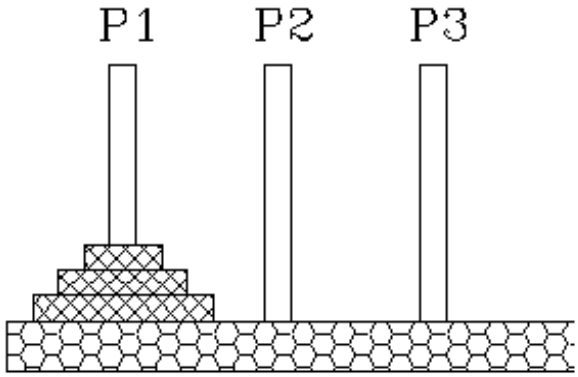
6 Erikoisalgoritmit

Tämä luku sisältää rekursiivisia funktioita sekä erikoisalgoritmeja. Lähes klassisena voisi pitää Hanoin tornit -ongelman ratkaisemista tietokoneella.

6.1 Hanoin tornit

Hanoin tornit muodostavat klassisen ongelman, jossa levyt on sijoitettava oikealla tavalla kolmeen eri pylvääseen. Levyt ovat pyöreitä ja niiden halkaisijat vaihtelevat. Levyjen keskellä on reikä pylväisiin sijoittamista varten. Vasemmassa pylväässä ovat aluksi kaikki levyt suuruusjärjestyksessä halkaisijan mukaan (suurin levy on alimpana).
Kuvassamme levyjä on 3 kappaletta:



Ongelma on nyt seuraavanlainen:
Kaikki levyt on siirrettävä vasemmasta pylväästä oikeanpuolimmaiseen pylvääseen siten, että

* vain yksi levy siirretään kerrallaan
* levyä ei saa koskaan sijoittaa pienemmän levyn päälle
* levyjen on oltava aina jossakin pylväässä (paitsi tietenkin siirrettävän levyn)

Kuinka monta siirtoa tarvitaan, on myös yksi kysymys, johon haetaan vastausta. Mutta tärkeintä on löytää siirtojärjestys.

Pylväitä on siis kolme kappaletta, joten siirtoja voidaan tukea käyttämällä jotakin pylvästä aina apupylväänä.

Esimerkiksi, jos levyjä olisi kaksi kappaletta, voidaan siirtää ensimmäinen levy keskimmäiseen pylvääseen ja sen jälkeen vasemman pylvään viimeinen (toinen) levy oikeanpuoleiseen pylvääseen. Lopuksi keskimmäisen pylvään levy siirretään oikeaan pylvääseen päällimmäiseksi.
Aina siis pätee se, että vasemman pylvään viimeinen levy siirretään tyhjänä olevaan oikeanpuoleiseen pylvääseen (muutoin sääntö 1 ei toteudu). Jos siis levyjä on X kappaletta, tulee X-1 kappaletta saada ensin keskimmäiseen pylvääseen käyttäen apuna oikeanpuoleista pylvästä ja sen jälkeen taas keskimmäisen pylvään levyt oikeanpuoleiseen pylvääseen käyttäen apuna vasemmanpuoleista pylvästä. Näin ongelma on hyvä ratkaista rekursiivisesti.
Jos numeroimme pylväät kuten kuvassamme oikealta vasemmalle P1, P2, P3, niin ensin tehdävät siirrot voidaan tehdä algoritmilla:

Hanoin tornit:

```
#include <iostream.h>
void siirra (int kiekkoja, char V, char O, char K);
void main()
{
const char V = 'A';
const char K = 'B';
const char O = 'C';
int kiekkoja;
cout << "Montako kiekkoa on? ";
cin >> kiekkoja;
siirra (kiekkoja, V, O, K);

}

void siirra(int x, char V, char O, char K)
{
if (x > 0 ) {
siirra(x-1, V, K, O);
cout << "Siirrän pylväästä " << V << " pylvääseen " << O << "\n";
siirra(x-1, K, O, V);
                          }
}
```

Hanoin tornien ratkaisualgoritmin suoritusaika on pahimmassa tapauksessa $O(N^2)$, kun $N>0$.

6.2 Rekursiivisiä algoritmeja

Rekursiossa proseduuri tai funktio kutsuu itseään. Tällaista kutsua sanotaan rekursiiviseksi. Rekursion etuina on mm., että koodi on hyvin suppeaa ja tiivistä ja rekursiossa käytetään paikallisia muuttujia. Rekursio on lähes välttämätön joissakin ohjelmointiratkaisuissa, kuten binääripuun käsittelyssä.

6.2.1 Kertoma rekursiolla

Kertoma on peräkkäisten, järjestyksessä olevien kokonaislukujen tulo. Kertoman merkkinä käytetään matematiikassa yleisesti huutomerkkiä (!). Esimerkiksi kertoma 3! = 1*2*3 = 6. Luku nolla (0) lasketaan kokonaislukuihin ja sen kertoma (0!) on yksi.

kertoma(0) = 1
kertoma(n) = n * kertoma(n-1); n>0

Kertoman algoritmi rekursiomuodossa näkyy seuraavassa ohjelmalistauksessa.

Kertoma (rekursio):

```
#include <iostream.h>
int kertoma(int n);

void main()
{
int a;

cout << "Minkä luvun kertoma lasketaan? \n";
cin >> a;

cout << "Kertoma on " << kertoma(a) << "\n";
}

int kertoma(int n)
{
int k;
if (n == 0) return(1);
else
return(n * kertoma(n-1));
}
```

Huomautus
Tarkista, että kertoman arvo sopii tietotyypin arvoalueelle. Käytä tarvittaessa vaikkapa double-tyyppiä suurentaaksesi arvoaluetta.

Rekursion toiminta
Jokaisessa kutsussa varataan parametrille muistitila, joten parametrien ilmentymiä on useita samaan aikaan (niin myös paikallisten muuttujien ja funktion ilmentymiä). Tuo itsensä kutsuminen tapahtuu ikään kuin toisesta funktiosta, joka on alkuperäisen funktion toinen, muistissa oleva ilmentymä.

Ajossa mennään nollaan saakka (n = 0), jonka jälkeen puretaan syntyneet funktioilmentymät; ne lasketaan viimeisestä ensimmäiseen seuraavassa simuloinnissa kuvatulla tavalla:

Jos x = 4
1. funktioilmentyma on 4 * kertoma(3)
2. funktioilmentyma on 3 * kertoma (2)
3. funktioilmentyma on 2 * kertoma (1)
4. funktioilmentyma on 1* kertoma (0)
   kertoma (0) on 1, jonka jalkeen puretaan (koska if-ehto tosi):
   kohdasta 4 tulee 1 * 1, tulos on 1 (kertoma (0) oli siis 1)
   kohdasta 3 tulee 2 * 1, tulos on 2 (kertoma (1) oli siis 1)
   kohdasta 2 tulee 3 * 2, tulos on 6 (kertoma (2) oli siis 2)
   kohdasta 1 tulee 4 * 6, tulos on 24 (kertoma (3) oli siis 6)

6.2.2 Fibonaccin luvut

Fibonaccin luku muodostetaan peräkkäisistä kokonaisluvuista. Luku on aina summa kahdesta edellisestä Fibonaccin luvusta. Fibonaccin luvut muodostavat sarjan, jonka peräkkäisten jäsenten suhde supistuu kohti ns. kultaisen leikkauksen suhdelukua (noin 1.612).

Esimerkiksi 9 ensimmäistä Fibonacci-lukua ovat seuraavat:

Järjestys:	1	2	3	4	5	6	7	8	9
Fibonacci-luku:	1	1	2	3	5	8	13	21	34

Ensimmäiset kaksi Fibonacci-lukua on määritelty ykkösiksi.

Algoritmi, jolla saadaan tietyssä kohdassa oleva Fibonacci-luku, on seuraavanlainen: Ensiksi tarkistetaan, onko annettu kohta järjestyksessä 1. tai 2. Fibonaccin luku, jolloin sen arvo on siis 1.

Fibonaccin luvut:

```
#include <iostream.h>
int fibonacci(int n);

void main()
{
int a;
cout << "Monesko fibonacci-luku haetaan? \n";
cin >> a;
cout << "Fibo on " << fibonacci(a) << "\n";
}

int fibonacci(int n)
{
    int arvo;
    if ( n== 1 || n== 2)
        arvo = 1;
    else
        arvo = fibonacci(n-1) + fibonacci(n-2);
    return(arvo);
}
```

Fibonaccin luvut muodostavat siis jonon. Muitakin jonoja on kätevää muodostaa rekursiolla. Yleisesti, jotta jokin lukujono olisi määritelty, on tunnettava sääntö, jonka mukaan voidaan muodostaa jonon yleinen jäsen k_n . Jono voidaankin määritellä kuvaamalla yleisen jäsenen k_n määrittely. Rekursiossa annetaan yleensä muutama alkuarvo, joiden perusteella aletaan laskea muita arvoja.

Aritmeettinen jono
Aritmeettisessa jonossa kahden peräkkäisen jäsenen *erotus* on vakio.
Siis $k_n - k_{n-1}$ = vakio c. Jäsenet ovat tällöin positiivisia kokonaislukuja.

Jos aritmeettisen jonon ensimmäinen jäsen on k ja peräkkäisten jäsenten erotus c, voidaan jono muodostaa rekursiolla seuraavasti:

$k_1 = k$
$k_{n+1} = k_n + c$
(tai $k_n = k_{n-1} + c$)

Jokin jonon jäsen (esimerkiksi 11. jäsen) haetaan lausekkeella:

$k_n = k + (n-1)c$

Geometrinen jono

Geometrisessa jonossa kahden peräkkäisen jäsenen suhde on vakio q. Eli $k_n / k_{n-1} = q$. Jos ensimmäinen jäsen $k_1 = k$ ja suhdeluku on q, saadaan:

$k_{n+1} = qk_n$

Ja yleinen jäsen on $k_n = kq^{n-1}$

6.2.3 SYT

Kahden positiivisen kokonaisluvun suurin yhteinen tekijä (SYT) saadaan seuraavasti:

syt(a,b) = a, jos b = 0
= syt(b, a % b) muulloin (a % b on a modulo b)

Suurin yhteinen tekijä (SYT)

```
#include <iostream.h>
int syt(int a, int b);
void main()
{
    int a,b,s;
    cout << "Anna 2 positiivista kokonaislukua\n";
    cout << "Anna 1. kokonaisluku\n";
    cin >> a;
    cout << "Anna 2. kokonaisluku\n";
    cin >> b;
    if (a >= b) s = syt(a, b);
        else s = syt(b, a);
    cout << "\nSYT = " << s;
}

int syt(int a, int b)
{
    if (b) return syt(b , a % b);
        else return a;
}
```

6.2.4 Lukujen summa

Lukujen summa lasketaan seuraavassa rekursiivisesti. Yhteenlaskettavia positiivisia kokonaislukuja voi olla n kappaletta.

Lukujen summa (rekursio):

```
#include <iostream.h>

void main()
{
    int summa(int luku), lkm;
    cout << "Montako lukua lasketaan yhteen: ";
    cin >> lkm;
    cout << "Lukujesi summa = " << summa(lkm);
}
    int summa(int luku)
    {
        if (!luku) return luku;    /* Palauttaa nollan */
        else return luku + summa(luku-1);
    }
```

Jos luvut ovat taulukossa, saadaan summa algoritmilla:

```
int summa(int a[], int ind)
{
    if (!ind) return a[ind];
        else return a[ind] + summa(a,ind-1);
}
```

6.2.5 Tyhjien alkumerkkien poistaminen merkkijonosta

Merkkijono-operaatioita käsittelevässä luvussa on algoritmi tyhjien alkumerkkien poistamiseen. Tämä voidaan toteuttaa myös rekursiivisesti, kun käytössä on C++-työkalu, joka tukee uusinta standardiehdotusta. Uudessa standardiehdotuksessa on string-luokka, joka sisältää muun muassa metodin remove(n, k).
Metodi remove(n, k) poistaa merkkijonosta n merkkiä alkaen kohdasta k.
Kyseinen string-luokka saadaan C++-ohjelmaan komennoilla

#include <string>

Nyt voisimme kirjoittaa rekursiivisen algoritmin tyhjien alkumerkkien poistamiseen.

Tyhjien alkumerkkien poistaminen merkkijonosta:

```
poistatyhjat(string s)
if (s[0] = ' ')
s.remove(1,0)  // poistetaan tyhjäksi merkiksi havaittu merkki
    poistatyhjat(s)
```

Samanlaisella menettelyllä voisimme poistaa myös tyhjät loppumerkit. Tällöin on vain otettava selville merkkien lukumäärä merkkijonossa ja poistettava aina viimeinen merkki ennen uutta funktiokutsua.