11 Muunnokset ja tekniset laskelmat

Tämä luku sisältää suppeita muunnoksia ja teknisiä laskelmia. Mukana on myös ASCII-koodi sekä lukujärjestelmien käsittely.

11.1. Celsius-asteet Fahrenheit-asteiksi

Celsius-asteina oleva lämpötila muunnetaan Fahrenheit-asteiksi johtamalla vastaava kaava käyttäen apuna termodynaamista lämpötilaa. Termodynaaminen lämpötila (T) ilmoitetaan Kelvin (K) -asteina, jolloin $T_0 = 273{,}15$ K. T_0 on nolla Celsius-astetta (t_c) ja vastaavasti 32 Fahrenheit-astetta (t_f). Sata Celsius-astetta on 373 K-astetta ja 212 Fahrenheit-astetta.

Siis kaavoina:

$$T = T_0 + t_c$$
$$T = (212-32)/373^* (t_f - 32) + T_0 = 5/9 * (t_f - 32) + T_0$$
$$1t_f = 9/5 * t_c + 32$$

Vastaavasti Fahrenheit-lämpötilan muuntaminen Celsius-asteiksi tapahtuu kaavalla:

$$t_c = 5/9^* (t_f - 32)$$

Muunnos fahrenheit->celsius:

```
float fahrenheit(int celsius)
{
    return(9.0/5.0 * celsius + 32);
}
```

11.2. Mittayksiköiden muunnokset yleensä

Mittayksiköiden muunnokset esimerkiksi C-kielellä tehdään kätevimmin aivan yksinkertaisesti määrittelemällä makroja #define-lauseella.

Seuraavana on pari mallia, joiden tarkkuuden voit itse määritellä käyttämällä sopivaa desimaalimäärää:

Muunnosmalleja:

Tuumat millimetreiksi:

```
#define TUUMA_MM(tuuma) tuuma * 25.4
```

(Vastaavasti millimetrit tuumiksi kertoimella 1/25.4)

Radiaanit asteiksi:

```
#define RAD_ASTE(aste) aste * 57.29578
```

C++-kielen inline-funktio olisi kuitenkin turvallisempi vaihtoehto. Inline-funktion rajoituksena saattaa olla se, että se näkyy vain oman lähdekooditiedostonsa sisällä static-funktioiden tapaan.

Seuraavana on esimerkki #define-makron vaarallisuudesta. Makro laskee neliön.

```
#include <iostream.h>

#define nelio(x) x*x

void main()
{
int a = 3;
cout << nelio(a) << "\n";              // tulostaa 9, OIKEIN
cout << nelio(a+1) << "\n";            // tulostaa 7, VÄÄRIN

}
```

Kun lisäämme makron esittelyyn sulkumerkit, saamme oikeat tulokset:

```
#define nelio(x) (x)*(x)
```

Nyt lause

```
cout << nelio(a+1) << "\n";
```

tulostaa 16, kuten pitääkin.

Vastaava toteutus inline-funktiolla:

```
#include <iostream.h>

inline int nelio(x){return x*x;};

void main()
{
int a = 3;
cout << nelio(a) << "\n";     // tulostaa 9
cout << nelio(a-1) << "\n";   // tulostaa 16
}
```

11.2.1 ASCII-koodit

ASCII-koodeja tarvitaan usein. Esimerkiksi näppäimistöltä halutaan silloin tällöin saada esille jokin merkki, jota ei ehkä löydy tekstinkäsittelyversiosta tai itse näppäimistö ei sisällä merkkejä. Myös tietokone voi olla jokin muu kuin PC-tietokone, esimerkiksi sellainen prosessinohjaustietokone, joka tunnistaa ASCII-koodiston. Usein ASCII-taulukkoa ei kuitenkaan ole käsillä silloin, kun sitä tarvittaisiin. Käyttämällä apuna pientä työkaluohjelmaa, johon seuraavassa esitetään algoritmi, päästään ongelmasta. Algoritmi antaa luvun, jolla yhdessä ALT-näppäimen kanssa saadaan tulostettua haluttu merkki.

ASCII-lyhenne tulee sanoista American Standard Code for Information Interchange. Koodiston avulla annetaan tietokoneen käyttämät numerot, kirjaimet ja eräät muut merkit. Onkin tärkeää, että erilaiset tietokoneet voivat ymmärtää ASCII-muotoisia merkkejä, koska usein tietoa joudutaan siirtämään tietokoneympäristöstä toiseen. ASCII-koodistossa on jokaiselle merkille annettu oma numeroarvonsa. Kukin merkki vie tilaa 8 bittiä eli yhden tavun. Tällaisen muistipaikan sisältö on siis se numeroarvo (binäärimuodossa), joka koodistossa on merkille määritelty. Esimerkiksi huutomerkille (!) on määritelty ASCII-koodi numero 32 (desimaalimuodossa). Huutomerkki tallennetaan siis muistipaikkaan binäärikuviona 00100000.

Joissakin C-kielen komennoissa voidaan numeroarvoa käyttää suoraan muuttujana. Tällainen on esimerkiksi putchar(). Lause putchar(32) tulostaisi siis huutomerkin (!).

Kuten jo tiedät, on numeron nolla (0) ASCII-koodi 48. Numeron yksi (1) ASCII-koodi on 49 ja niin edelleen. ASCII-merkkijonon merkkiä vastaava kokonaisluku saadaan siis aivan yksinkertaisesti vähentämällä annetusta ASCII-koodista nollaa vastaava ASCII-koodi (katso allaolevia ohjelmalauseita).

```
char a = '5';
cout << "Merkin " << a  << " ascii on " << (int)a;
cout << " ja merkki on numerona " << ((int)(a) - (int)('0'));
```

ASCII-merkkijono desimaalinumeroiksi:

```
#include <iostream.h>

int muunna(char ascii_arvo[], int pituus);
void main()
{
    int asluku, koko;
    char luvut[] = "230";
    asluku = muunna(luvut, 2);
    cout << asluku;
}

int muunna(char ascii_arvo[], int pituus)
{
    int paikka;      /*sijainti merkkijonossa*/
    int arvo = 0;        /*haettu kokonaislukuarvo*/

    for (paikka=0; paikka<pituus; paikka++)
    if(ascii_arvo[paikka] <= '9' && ascii_arvo[paikka] >= '0')
        arvo = arvo * 10 + ascii_arvo[paikka] - '0';
    else
    {
        return(-1);
        break;
    }
    return(arvo);
}
```

Lisää ylle merkkijonon syöttötoiminto. Samoin on lisättävä syötön tarkistus, koska yllä oleva algoritmi muuntaa vain positiivisia kokonaislukuja sisältäviä merkkijonoja. Myös tiedoston loppumerkki (EOF) pysäyttää ohjelman, mikä on estettävä. Mikäli merkki saa esittää myös negatiivista kokonaislukua, on funktiota muutettava havaitsemaan miinusmerkki '-' ja ohittamaan se muunnosvaiheen ajaksi. Etumerkki huomioidaan vasta muunnoksen jälkeen (ehkä arvoa palautettaessa). Ohjelmaa tulee täydentää myös poistamalla merkkijonon tyhjät merkit sekä ilmoittamaan vääristä merkeistä.

Eräissä ohjelmointikielissä on suoraan muunnosfunktio, jolla saadaan merkin ASCII-järjestysnumero.

C++ -kielessä taas char käsitetään numeroarvona, joka tulee ascii-taulukosta. Voimme tällöin muuntaa ulkoisesti int-tyypin char-tyypiksi.

Esimerkiksi:

```
char m = 'a';
cout << "Merkin " << a << " ascii-koodi on " << (int)a;
```

Numeroarvo on nyt siis ascii-koodi, josta tulee vähentää arvo 48, jotta todellisen numeroarvon saa esille.

Huomautus
Unicode-koodisto on 16-bittinen koodisto ja esimerkiksi Java käyttää sitä merkkien koodaukseen. Unicoden alkuosassa on perinteinen Ascii-koodisto. Unicode-koodistoon mahtuu siis 2 ^16 eri koodia eli 65536 kpl. Ascii-koodiston perusjoukko on 7-bittinen ja Euroopassa yleisesti käytetty laajennus on nimeltään Latin-1. Kaikkien kielten merkit eivät mahdu 8-bittiselle alueelle, joten Unicode parantaa tilannetta. Unicoden ongelmana (ja etuna) on juuri tuo 2-tavuisuus johtuen tavujen tallennusjärjestyksen vaihtelusta eri ympäristöissä: niinpä onkin kehitetty koodaustapa nimeltä UTF-8, johon Unicode-merkit käännetään siirtoa varten. UTF-8 muuntaa tavut 1-, 2- ja 3-tavuisiksi merkeiksi riippuen niiden paikasta Unicode-koodistossa.

11.2.2 Aikayksikkömuunnos

Sekunneissa annettu aika voidaan muuttaa tunneiksi, minuuteiksi ja sekunneiksi seuraavalla algoritmilla:

Sekunnit tunneiksi ja minuuteiksi:

```
#include <iostream.h>

void main()
    {
    int aika, tunnit, minuutit, sekunnit;
    // Annetaan sekunnit

    cout << " anna sekuntien määrä \n";
```

```
    cin >> aika;

    // Muunnokset
    tunnit = aika / 3600;  // Kokonaislukujakolasku
    minuutit = (aika % 3600) / 60;
    sekunnit = aika % 60;
    // TULOSTUS

    cout << " Tunteja on: " << tunnit << "\n";
    cout << " Minuutteja on: " << minuutit << "\n";
    cout << " sekunteja on: " << sekunnit << "\n" ;
}
```

11.2.3 Rahasumma rahayksiköiksi

Markkoina annettu rahamäärä voidaan muuttaa seteleiksi ja kolikoiksi seuraavalla algoritmilla.

Rahamuunnos:

```
#include <iostream.h>

void main()
    {
    long    s1000, s500, s100, s50, s20, loput;
    // Annetaan setelit

    // s merkitsee setelimääriä ja k, kp kolikkomääriä
    long summa;
    cout << "anna markkamäärä ";
    cin >> summa;
    s1000 = summa / 1000;
    s500 = (summa % 1000) / 500;
    s100 = (summa % 500) / 100;
    s50 = (summa % 100) / 50;
    s20 = (summa % 50) / 20;
    loput = summa % 20;

    cout << " Tonnit: " << s1000 << "\n";
    cout << " Viissataset: " << s500 << "\n";
    cout << " Sataset: " << s100 << "\n" ;
    cout << " Viiskymppiset: " << s50 << "\n" ;
    cout << " Kakskymppiset: " << s20 << "\n" ;
    cout << " Kolikoita: " << loput << "\n" ;
}
```

Simuloimme tätä esimerkillä:

Olkoon summa: 7830 mk
1. 7830 / 1000 = 7 * 1000 mk
2. 7830 % 1000 = 830 / 500 = 1 * 500 mk
3. 7830 % 500 = 330 / 100 = 3 * 100 mk
4. 7830 % 100 = 30 / 50 = 0 * 50 mk
5. 7830 % 50 = 30 / 20 = 1 * 20 mk
5. 7830 % 20 = 10 / 10 = 1 * 10 mk

11.2.4 Kuukausien nimien ilmoittaminen

Joskus sovelluksiin halutaan laittaa kuukausien nimet havainnollisuuden lisäämiseksi. Käyttäjää ei kuitenkaan haluta rasittaa liiallisella syöttämistyöllä. Seuraava algoritmi pyytää käyttäjää antamaan päivämäärän muodossa PPKKVV, ottaa sitten merkkijonosta esille kuukausiluvun ja ilmoittaa vastaavan kuukauden nimen. Esimerkiksi annettu päivämäärä 120395 tulostetaan muodossa 12. maaliskuuta, 1995.

Päivämäärän muunto:

```
#include <iostream.h>

void main()
{
char pvm[6];
int merkki2, merkki3, kuukausi;
cout << "Anna päivämäärä (PPKKVV): ";
cin >> pvm;
merkki2 = (int)pvm[2] - (int)('0');  // Vähennetään nollan ascii
merkki3 = (int)pvm[3]- (int)('0');

kuukausi = 10*merkki2 + merkki3;

cout << kuukausi;
cout << "\n";
cout << pvm[0] << pvm[1] << "." ;

switch(kuukausi)
{
case    1: cout << "tammikuuta"; break;
case 2: cout << "helmikuuta"; break;
case 3: cout << "maaliskuuta"; break;
case 4: cout << "huhtikuuta"; break;
case 5: cout << "toukokuuta"; break;
case 6: cout << "kesäkuuta"; break;
case 7: cout << "heinäkuuta"; break;
case 8: cout << "elokuuta"; break;
case 9: cout << "syyskuuta"; break;
case 10: cout << "lokakuuta"; break;
case 11: cout << "marraskuuta"; break;
case 12: cout << "joulukuuta"; break;
}
cout << ", 19" << pvm[4] << pvm[5];
}
```

11.2.5 Lukujärjestelmämuunnokset

Yleisimmät *lukujärjestelmät* ovat:

desimaali: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
binääri: 0, 1
heksadesimaali: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
oktaali: 0, 1, 2, 3, 4, 5, 6, 7

Desimaalilukujen kantalukuna on 10, heksadesimaalilukujen 16, oktaalilukujen 8 ja binäärilukujen 2. Myös muihin kantalukuihin perustuvia lukujärjestelmiä käytetään joissakin sovelluksissa. Näissä viimeksi mainituissa tapauksissa muunnos tehdään yleensä kierrättämällä desimaalilukujen kautta eli ensin muunnettava luku muunnetaan desimaaliluvuksi, joka sen jälkeen muunnetaan toisen kantaluvun järjestelmään.

Katsotaan ensin esimerkkien valossa, kuinka muunnokset suoritetaan manuaalisesti:

Desimaaliluku heksadesimaaliluvuksi
Olkoon desimaalilukuna luku 339

339/16 = 21, jäljelle jää 3
21/16 = 1, jää 5
1/16 = 0, jää 1

Kyseessä oli kokonaisluku, joten (onneksi) desimaaleja ei tarvitse määrittää.

Siis: $339_d = 153_h$

Tarkistus tehdään suorittamalla muunnos heksadesimaaliluvusta desimaaliluvuksi.

$1* 162 + 5* 161 + 3 * 160 = 339$

Desimaaliluku muunnetaan oktaaliluvuksi samalla periaatteella. Ero on vain kantaluvussa, joka on siis kahdeksan.

Desimaaliluku binääriluvuksi:
Olkoon luku vaikkapa 95:

95/2 = 47 jää 1
47/2 = 23 jää 1
23/2 = 11 jää 1

11/2 = 5 jää 1
5/2 = 2 jää 1
2/2 = 1 jää 0
1/2 = 0 jää 1

Siis $95_d = 1011111_b$

Tarkistus tehdään tekemällä päinvastainen muunnos:

$1011111 = 1*2^6 + 0*2^5 + 1*2^4 + 1*2^3 + 1*2^2 + 1*2^1 + 1*2^0 = 95$

Muunnos desimaaliluvusta eri kantaluvussa olevaan muotoon voi tapahtua esimerkiksi seuraavasti:

Lukujärjestelmämuunnokset:

```
#include <iostream.h>

void desimuunnos(int n, int kanta);

void main()
{
int desi;
int kanta;

cout << "Anna positiivinen desimaalikokonaisluku: ";
cin >> desi;

desimuunnos(desi,2);

}

void desimuunnos(int n, int kanta)
{
const int maksimi = 10;
char uusluku[] = {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9',
'A', 'B', 'C' ,'D', 'E', 'F'};
int i, jaannos;
char merkki[maksimi];

i = 0;

do
{
    i = i+1;
    jaannos = n % kanta;
```

```
        merkki[i] = uusluku[jaannos];
        cout << merkki[i] << "\t";
        n = n / kanta;
    }
    while (n > 0);
    }
```

Myös reaaliluku, jossa on desimaaliosia, voidaan muuntaa eri lukujärjestelmämuotoihin. Tällöin desimaaliosan kymmenykset kerrotaan kantaluvulla ja tulon kokonaisosa tulee muunnetun luvun ensimmäiseksi desimaaliosaksi. Loppuosa kerrotaan taas kantaluvulla ja tehdään vastaavalla tavalla. Usein pyritään tietenkin siihen, että myös muunnetussa luvussa on yhtä paljon (pyöristyksen jälkeen) desimaaleja kuin alkuperäisessä luvussakin (edellyttäen tietenkin, että desimaaliosa on päättymätön).

Sovellus: RGB-väri heksadesimaalimuotoon

Tietokoneessa esitetään värit RGB-muodossa (R=Red, G=Green, B=Blue). Kukin värin komponentti voi saada yhden tavun bittejä eli komponentin arvo voi vaihdella väliltä 0 – 255. Joissakin tapauksissa kuten esimerkiksi www-sivuilla (html-koodi) halutaan värin olevan yleensä heksadesimaalimuodossa. Pienellä sovelluksella voit tehdä muunnoksen; muunnos on nyt yksinkertainen, koska maksimiluku on 255 (eli heksadesimaalina FF).
Heksadesimaaliluvun suuremman arvon saat generoimalla indeksin heksamerkkitaulukkoon kokonaislukujakolaskulla ja pienemmän arvon taas jakojäännöksen avulla. Seuraava esimerkki valaisee asiaa:

```
char heksataulu[] = “0123456789ABCDEF”;
unsigned short red = 200;
char heksa[2];
unsigned short indeksi1 = red / 16;
unsigned short indeksi2 = red % 16;
heksa[0] = heksataulu[indeksi1];
heksa[1] = heksataulu[indeksi2];
```

Jos haluat heksadesimaalimuotoiseen väriin mukaan kaikki värikomponentit, voit antaa heksa[]-taulukolle kooksi 6 ja sijoittaa saadut heksamerkit oikeassa järjestyksessä (RGB) taulukkoon.

Seuraavana on vielä esimerkki positiivisen kokonaisluvun tulostamisesta binäärimuodossa. Käytetty funktio on rekursiivinen.

Binäärimuoto rekursiivisesti:

```
#include <iostream.h>

void bin(int i)
{
  if (i>0)
  {
     bin(i / 2);
     cout << i % 2;
  }
}


main()
{
int a = 9;

cout << " binaarina on ";
bin(a);

return 0;
}
```

Entä, jos laittaisit binääriarvot taulukkoon, josta ne voisi tulostaa esimerkiksi käänteisessä järjestyksessä? Voisimme asettaa ylärajan muunnettavalle kokonaisluvulle ja päättää sen avulla tarvittava taulukon koko. Toinen mahdollisuus olisi siirtää muunnettavaa kokonaislukua (apumuuttujan avulla) bittitasolla oikealle yksi askel kerrallaan, kunnes luku on yksi. Askelten määrä kertoo sitten taulukossa tarvittavien alkioiden määrän.

Seuraavat lauseet antavat bittijonon koon:

```
int a = 9; int b = a;
for(int i = 1; b != 1; i++)
b = b >> 1;
const int KOKO = i;  // Sijoitetaan vakioon
```

Otamme vielä esille yhden tietotekniikassa käytetyn koodausmenetelmän, joka on nimeltään *BCD-koodaus*. BCD-lyhenne tulee sanoista Binary Coded Desimal. Seuraavassa luettelossa on binäärimuodot luvuista 0 ... 9, ja niiden avulla muodostetaan suurempien desimaalilukujen BCD-koodit.

0 0000
1 0001
2 0010

3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

BCD-koodi muodostetaan siten, että desimaaliluvun kukin numeroarvo saa erillisen binäärikoodinsa, joista sitten koko BCD-koodinen luku muodostuu.

Esimerkiksi luvun 233 BCD koodi on

0010	0011	0011
2	3	4

Kutakin arvoa väliltä 0 ... 9 esittää siis nelinumeroinen binääriluku. Yleensä loppupään nollat jätetään pois, ainakin käytettäessä ns. pakattua BCD:tä.

Algoritmin voi laatia erottelemalla annetun desimaaliluvun ykkösiä, kymmeniä, satoja jne. kuvaavat luvut (0...9) ja esittämällä niitä vastaavat BCD-koodit oikeassa järjestyksessä. Tässä tapauksessa kannattanee laittaa numeroiden 0 ...9 vastaavat BCD-koodit taulukkoon, jolloin säästytään laskemiselta.

Huom! BCD-koodista saadaan suuri binääriluku muunnettua heksadesimaalimuotoon erittäin kätevästi. Kun suuri binääriluku jaetaan neljän bitin ryhmiin, voidaan heksadesimaaliarvo määrätä kullekin ryhmälle erikseen (suurinta heksadesimaaliarvoa F ei siis koskaan ylitetä, koska neljän bitin edustama maksimiarvo on aina <=16). Kun arvosta 0 ei välitetä, saadaan koko binärilukua vastaava heksadesimaaliluku asettamalla ryhmiä vastaavat heksadesimaaliluvut peräkkäin.