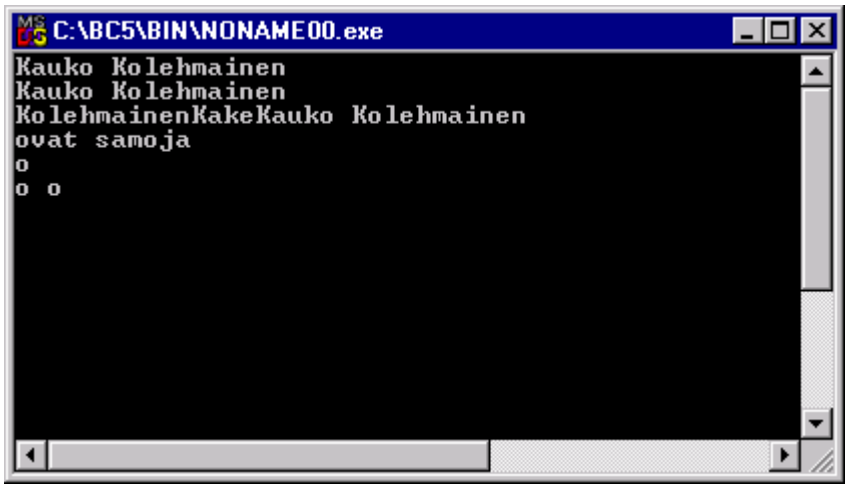
# 3 Merkkijonoalgoritmit

Tässä luvussa käsitellään merkkijonojen manipulointia. Mukana on melko suppeitakin ratkaisuja. C++-standardissa on string-luokka, jonka avulla merkkijonojen käsittely on aiempaa mukavampaa. Joissakin sovelluksissa joudutaan kuitenkin tinkimään kaikista ylimääräisistä koodiriveistä, joten myös itse kirjoitettuja merkkijonofunktioita tarvitaan.

String-luokan kautta voidaan Javan tapaan muodostaa merkkijono-olioita (string-olioita). Luokka sisältää runsaasti metodeja ja operaattoreita.

### Esimerkki: string-luokan käyttö

```
#include <iostream.h>
#include <conio.h>
#include <cstring.h>           // TAI #include <string>
 main ()
 {
   string a1 = "Kauko";
   string a2, a3;
   a2 = a1;
   a2 += " Kolehmainen";
   a3 = "Kake" + a2;
   a1.insert(5, " Kolehmainen");
   a3.insert(0, a1, 6, 11);
   cout << a1 << endl;
   cout << a2 << endl;
   cout << a3 << endl;
   if (a1 == a2)
     cout << "ovat samoja " << endl;
   a1 = a2.substr(1, 5);
   if (a1 < a2)
     cout << "On ennemmin " << endl;
   a1.remove(0, 3);
   a2 = a1 + a3[1];
   cout << a1 << endl;
   cout << a2 << endl;


  getch();
}
```




# 3.1 Merkin paikka merkkijonossa

Tämä algoritmi hakee merkin ensimmäisen sijaintipaikan merkkijonossa. Jos merkkiä ei esiinny merkkijonossa, palautetaan arvo -1. Merkkijono tulee olla määritelty ensin ja sen on päätyttävä loppumerkkiin \0.

Merkkijono voidaan C++-kielessä esitellä merkkitaulukkona tai char-tyyppisen osoittimen avulla:

```
char taulu[10];
char *merkkijono;
```

### Merkin sijainti merkkijonossa:

```
#include <iostream.h>

void main()
{
    char mj[] = "Tilaus";   // sisältää merkkijonon
    char m;      // etsittävä merkki
    int paikka;    // paikka merkkijonossa //
    int on;
    m = 'a';

    for (paikka = 0; mj[paikka] != NULL ; paikka++)
        {
        if ( mj[paikka] == m )
            {
            on = 1;
```

```
            break;
            }
        on = 0;    // merkkiä ei esiintynyt merkkijonossa //
        }


    if (on == 1) cout << paikka;
    else cout << "Ei ole";

}
```

### Huomautus

Otsikkotiedosto string.h sisältää funktion strchr(), joka antaa merkkijonon paikan. Myös string-luokka sisältää metodeja merkkijonon merkkien käsittelyyn.

## 3.2 Merkkien lukumäärä merkkijonossa

### Merkkien määrä merkkijonossa:

```
#include <iostream.h>

void main()
{
    char mj[] = "Tilaus";   // sisältää merkkijonon
    int lkm;   /*merkkien lukumäärä merkkijonossa*/
    /* nyt lasketaan merkkien lukumäärä aina
    loppumerkkiin saakka*/
    for (lkm = 0; mj[lkm] != NULL; ++lkm);
    cout << lkm;
}
```

### Huomautus

Otsikkotiedosto string.h sisältää funktion strlen(), joka antaa merkkijonon pituuden.

# 3.3 Merkkijonon esittäminen käänteisessä järjestyksessä

### Merkkijonon kääntäminen:

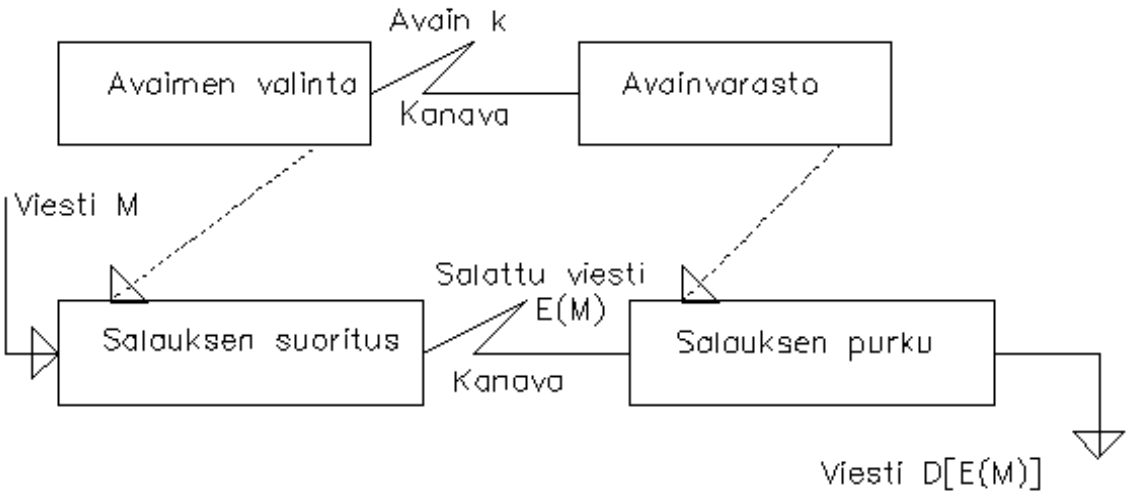
```
#include <iostream.h>

void main()
{
    char mj[] = "Tilaus";   // sisältää merkkijonon
    int lkm, paikka;   /*merkkien lukumäärä merkkijonossa*/
    cout << "merkkijono on "<< mj << "\n";
    /* nyt lasketaan merkkien lukumäärä aina LOPPU-merkkiin saakka*/
    for (lkm = 0; mj[lkm] != NULL; ++lkm);
    cout << "merkkejä on " << lkm << " kpl\n";

    cout << "merkkijono käännettynä on: \n";
/* mj[lkm] on nyt loppumerkin suuruinen, joten aloitetaan
kääntäminen alkaen paikasta lkm-1 */
    for (paikka = lkm-1; paikka>=0; --paikka)
        cout << mj[paikka];
        cout << "\n";
}
```

### Huomautus

Myös strrev() kääntää merkkijonon.

# 3.4 Krypteeraus eli salakirjoitus

Pelkistetty salausjärjestelmä voidaan kuvata seuraavalla kaaviolla:

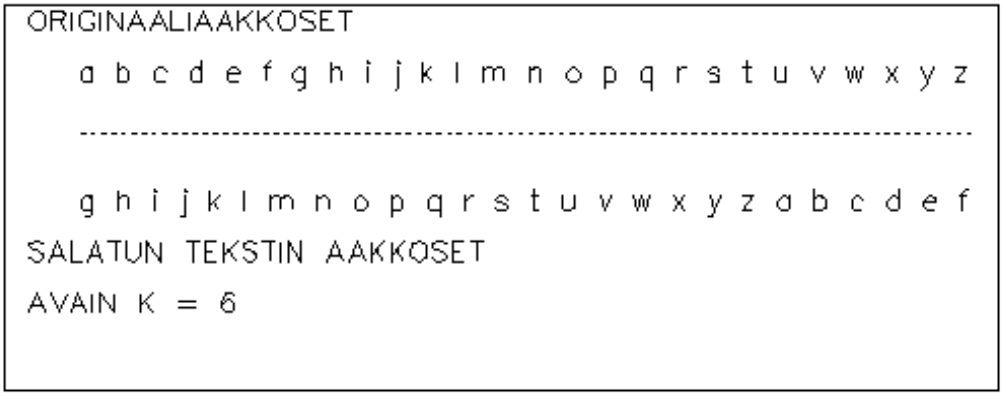



Selväkielinen teksti M muunnetaan salatuksi tekstiksi E(M), jonka tekstin vastaanottaja muuntaa takaisin selväkieliseksi tekstiksi muunnoksella M = D[E(M)]. Salaus sisältää siis algoritmin ja avaimen. *Symmetrisessä salauksessa* käytetään samaa avainta sekä salaukseen että salauksen purkuun, kun taas *epäsymmetrisessä salauksessa* salausavain on eri kuin purkuavain. Symmetristä salausta käytettäessä tulee siis avaimen pysyä salaisena, jolloin se on siirrettävä vastaanottajalle turvallista kanavaa pitkin.

## 3.4.1 Klassiset menetelmät

Ensimmäiset salakirjoitus- (krypteeraus-) menetelmät perustuivat siihen, että selväkielisen tekstin kirjaimet korvattiin toisilla kirjaimilla, jotka saatiin esimerkiksi siirtymällä aakkosissa tietty kirjainväli. Tällaista *korvausalgoritmia* kutsutaan nimellä Caesarin salausalgoritmi.

Seuraava esimerkki kuvaa korvaavan salausmenetelmän käyttöä:




Esimerkkimme avaimena K on 6, jolloin kirjainta a vastaa salatussa tekstissä kirjan g jne. Esimerkiksi viesti 'ammukset lopussa' olisi tällöin salattuna 'gssaqykz ruvayyg'. Salatun tekstin selvittäminen ei tällaisessa tapauksessa ole mahdottoman hankalaa, mutta on muistettava, ettei tietotekniikkaa voitu aiemmin hyödyntää ja toisaalta kysymys oli usein ajasta, joka selvittämiseen kului silloin, kun salattu teksti joutui väärälle osapuolelle. Avaimen K kuljettamiseen tarvittiin usein kuriiria.

Useimmiten korvausmenettelyn laskenta oli tietenkin huomattavasti monimutkaisempi. Tällaisen salakirjoituksen heikkoutena on nykyään se, että useimmissa kielissä eri kirjainten esiintymistaajuus tekstissä on tilastollisesti pääteltävissä. Tällöin täytyy analysoida myös salattua tekstiä kirjaintaajuuksien suhteen (ongelmana saattaa kuitenkin olla se, että salattu viesti voi olla niin lyhyt, ettei sitä voida tilastollisesti tutkia). Esimerkiksi suomen kielessä ovat a-ja t-kirjaimet yleisimpiä. Tuolta pohjalta voitaneen päätellä, mitä 'oikeaa' kirjainta tietty salakirjoitustekstin kirjain vastaa.

Edellistä menettelyä parempi salaus saadaan käyttämällä salaista kirjainjoukkoa, jolla korvataan selväkielisen tekstin kirjaimet järjestyksessä alusta alkaen. Kirjainjoukolla korvaaminen toistetaan, kunnes koko teksti on salattu. Tällä menettelyllä vältetään kielellisten ominaisuuksien aiheuttamat ongelmat.

Seuraavassa algoritmissa merkkijono salakirjoitetaan toiseen merkkijonoon käyttämällä salakirjoitusavaimen yhtenä tekijänä merkin paikkaa merkkijonossa.

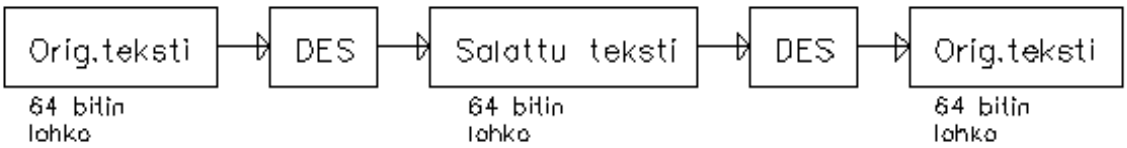
**Salakirjoitus:**

```
#include <iostream.h>

void main()
{
char* selvateksti;
char koodattuteksti[]="";
int i, lkm;
char uusimerkki;

cout << "Anna merkkijono: \n";
cin >> selvateksti;

for (lkm = 0; selvateksti[lkm] != NULL; ++lkm);

for (i= 0; i < lkm; i++)
{
    uusimerkki = char(selvateksti[i] + 4);
    koodattuteksti[i] = uusimerkki;
}

cout << koodattuteksti;

}
```

## 3.4.2 Modernimmat salausmenetelmät

Otamme aiempia nykyaikaisemmat salausalgoritmit esille vain lyhyesti, koska niiden yksityiskohtaisempi käsittely tämän teoksen puitteissa veisi aivan liian paljon tilaa. Lisäksi algoritmit on usein suunniteltu toteutettavaksi piireillä.

*DES* (Data Encryption Standard) on 1970-luvulla kehitetty amerikkalainen menetelmä. Algoritmia voi tutkia tarkemmin esimerkiksi ANSIn standardista. DES-menettelyssä teksti jaetaan 64-bittisiin lohkoihin ja avain on 56-bittinen (lisäksi tulee 8 pariteettibittiä). Menetelmä on symmetrinen eli samaa avainta käytetään sekä salaukseen että salauksen purkamiseen, kuten allaoleva kuva esittää:



DES-menetelmässä pääavain luodaan usein satunnaislukugeneraattorilla. Salausprosessin aikana pääavaimesta luodaan 16 eri apuavainta. Prosessissa käytetään epälineaarista funktiota, jonka tekijöinä 16 eri vaiheessa ovat apuavaimet ja osa 64-bittisestä tekstilohkosta. Ydinkomponentti on ns S-laatikko, jonka avulla 6 peräkkäistä bittiä sovitetaan 4 bitiksi. Tärkein looginen funktio, jota käytetään, on XOR-funktio (modulo2).

DES-salauksen varmuutta kuvaa se, että jos käsillä on selväkielinen teksti, josta halutaan saada aikaan annettu salattu teksti, on oikea avain löydettävä $2^{56}$ mahdollisen avaimen joukosta. Kuitenkin 56-bittistä avainta pidetään hieman lyhyenä. Tärkeintä onkin pitää avain salaisena ja kuljettaa sitä turvattuja reittejä käyttäen.

*Julkisen avaimen menetelmät* ovat epäsymmetrisiä menetelmiä, eli niissä salaus- ja purkuavaimet ovat erilaiset. Salausavain on julkinen, kun taas purkuavaimen tulee olla vain vastaanottajan tiedossa. Purkuavainta ei saa olla mahdollista johtaa salausavaimesta. *RSA* (nimi tulee menetelmän tekijöistä Rivest, Shamir ja Adleman) on tärkein julkisen avaimen menetelmä. RSA perustuu vaikeuteen jakaa suuria kokonaislukuja niiden perustekijöihin.

Proseduurin pääpiirteet ovat seuraavat:

1. Viesti M, salattu teksti, avaimet ja salaus- sekä purkumuunnokset esitetään positiivisina kokonaislukuina.

2. Salausavain $K_s$ ja purkuavain $K_p$ muodostetaan seuraavasti:

2.1. Valitaan kaksi suurta (yli 100 numeroarvoa) satunnaislukua p ja q, jotka ovat alkulukuja.

2.2. Muodostetaan kaksi yhtäsuuruusoperaatiota:

$n = p*q$

$r = (p-1)*(q-1)$

2.3. Valitaan satunnaisesti luku e siten, että

$e < r$ ja

e ei sisällä r:n tekijöitä

2.4. Lasketaan kokonaisluku d siten, että

$e*d = 1 \bmod r = 1 \bmod (p-1)*(q-1)$

2.5. Muodostetaan purkuavain muodossa (d, n) ja salausavain (e, n)

3. Menetelmän vahvuus on vaikeudessa löytää d. Tällöin on jaettava jopa yli 200 numeroa sisältävä luku n tekijöihinsä.

4. Viesti M muunnetaan salatuksi tekstiksi C siten, että
$$C = M^e \bmod n$$

5.Viesti puretaan selväkieliseksi tekstiksi laskemalla
$$M = C^d \bmod n$$

Prosessista saadaan parempi kuva, jos simuloimme sen päävaihetta (2) pelkistetysti:

2.1. Valitaan alkuluvut p ja q seuraavasti:
p = 83
q = 89

2.2. Luvut n ja r lasketaan nyt seuraavasti:
n = p* q = 83 * 89 = 7387
r = (p-1)*(q-1) = 82 * 88 = 7216

2.3. Valitaan satunnaisesti luku e siten, että
e < r ( < 7216 ) ja
e ei sisällä r:n tekijöitä (katsotaan r:n tekijät; r = 16 * 11 * 41
otetaan e = 1031

2.4. Lasketaan kokonaisluku d siten, että
e * d = 1 mod r = 1 mod (p-1)*(q-1)
1031 * d = 1 mod 7216

Saamme d = 7.

Tietenkin e:n arvoksi on valittavana muitakin lukuja.

**Salaus- ja purkuavaimet:**
Salausavain on siis (1031, 7387) ja purkuavain (7, 7387).

Viesti M muunnetaan salatuksi tekstiksi C siten, että
$$C = M^e \bmod n$$

Viesti puretaan selväkieliseksi tekstiksi laskemalla
$$M = C^d \bmod n$$

# 3.5 Muita merkkijono-operaatioita

## 3.5.1 Tietyn kirjaimen esiintyminen rivillä

Seuraavana on algoritmi, jolla haetaan jonkin kirjaimen esiintymistaajuus tekstirivillä.
Algoritmia voi kehittää edelleen esimerkiksi jo aiemmin laaditun tekstiasiakirjan tutkimiseen. Voit myös lisätä lauseet, joilla tekstiä syötetään tiedostoon, jota algoritmilla sitten tarkastellaan.
Seuraavan algoritmin avulla voitaisiin näin tutkia ensin alkeellisesti salakirjoitettua tekstiä ja taulukoida kirjainten esiintymistaajuudet. Tämän jälkeen tutkittaisiin suurempia määriä selväkielistä tekstiä, jolloin saataisiin kirjainten yleiset esiintymistaajuudet. Ristiintaulukoimalla kyseisiä taulukoita voitaisiin päätellä, mikä salakirjoituskirjain korvaa kunkin oikean kirjaimen.

Algoritmi, jolla tutkitaan k-kirjaimen esiintymistä rivillä. Lisää tekstirivin syöttölauseet ja tulostus. Rivin paikalle voit lisätä joko valmiin tai syötettävän tekstitiedoston.

**Kirjaimen esiintymistaajuus:**

```
#include <iostream.h>

void main()
{
char rivi[80];
int i, lkm, maara;
char merkki;

cout << "Anna merkkijono: \n";
cin >> rivi;

cout << "Anna merkki, jonka lukumäärä selvitetään: \n";
cin >> merkki;

for (lkm = 0; rivi[lkm] != NULL; ++lkm);

for (i= 0; i < lkm; i++)
{
        if (rivi[i] == merkki) maara++;
}
```

```
cout << "Merkkijonossa " << rivi << " esiintyi \n";
cout << "merkki" << merkki << " " << maara << " kertaa";

}
```

Edellä esitettyä algoritmia hieman muokkaamalla voidaan aikaansaada korvausalgoritmi, jota voidaan käyttää esimerkiksi ASCII-tiedostojen yhteydessä. Tällöin esimerkiksi tab-merkki voidaan kätevästi muuntaa muuksi merkiksi sellaisissa tapauksissa, kun tiedostoa siirrettäessä vaaditaan merkkijonojen (tai muiden tietoyksiköiden) erottelumerkin vaihtamista. Korvattava ja korvaava merkki kannattaa esittää ASCII-koodilla. Huomattakoon, että useimmat tekstinkäsittelyohjelmat sisältävät tällaisen ominaisuuden. ASCII-koodistoa ei kuitenkaan käytetä kaikissa ympäristöissä!

## 3.5.2 Merkkijonon sisällön vaihtaminen

Edellä oli puhetta merkkien korvaamisesta toisilla merkeillä. Joskus voi eteen tulla myös tilanne, jossa merkkijonoja on vaihdettava keskenään. Algoritmi, jolla kahden merkkijonon sisältö vaihdetaan keskenään, on seuraava:

### Merkkijonon vaihtaminen:

```
#include <iostream.h>
void vaihda(char *(*m1), char *(*m2));
void main()
{
char* rivi1;
char* rivi2;
int i, lkm, maara;
char merkki;

cout << "Anna 1. merkkijono: \n";
cin >> rivi1;

cout << "Anna 2. merkkijono: \n";
cin >> rivi2;


// Vaihdetaan merkkijonomuuttujien sisällöt
cout << "1. mj on: " << rivi1 << "\n";
cout << "2. mj on: " << rivi2 << "\n";
```

```
vaihda(&rivi1, &rivi2);

cout << "1. mj on: " << rivi1 << "\n";
cout << "2. mj on: " << rivi2 << "\n";

}
void vaihda(char *(*m1), char *(*m2))
{
     char *temp;
     temp = *m1;
     *m1 = *m2;
     *m2 = temp;
}
```

## 3.5.3 Kahden merkkijonon vertailu

Merkkijonojen vertailua tarvitaan hyvin usein. Esimerkiksi jonkun nimen oikeellisuus voidaan tarkistaa tiedostosta vertaamalla oikeaan merkkijonoon. Hyvin usein sattuu, että käyttäjä antaa hieman poikkeavia yritysten tms. nimiä tiedostoihin, jolloin sama yritys voi esiintyä useallakin eri nimellä.

### Merkkijonojen vertailu:

```
#include <iostream.h>

int sama(char *m1, char *m2);

void main()
{
char* rivi1;
char* rivi2;
int i, lkm, maara;
char merkki;

cout << "Anna 1. merkkijono: \n";
cin >> rivi1;

cout << "Anna 2. merkkijono: \n";
cin >> rivi2;

int a = sama(rivi1, rivi2);

cout << "a on: " << a << "\n";

}
int sama(char *m1, char *m2)
```

```
{
    while (*m1 == *m2++)
    {
    if (*m1++ == NULL) return(1);
    }
    return(0);
}
```

## 3.5.4 Palindroma

Palindroma on sana, joka on samanlainen luettaessa oikealta tai vasemmalta. Esimerkiksi saippuakauppias on palindroma. Se, onko jokin sana palindroma, voidaan tarkistaa helposti silmukassa (sanan pituus olkoon p ja merkkijono s[]):

**Palindroma:**

```
#include <iostream.h>

void main()
{
    char* mj;   // sisältää merkkijonon
    int lkm, i;   /*merkkien lukumäärä merkkijonossa*/
    cout << "Anna merkkijono \n";
    cin >> mj;
    /* nyt lasketaan merkkien lukumäärä aina LOPPU-merkkiin saakka*/
    for (lkm = 0; mj[lkm] != NULL; ++lkm);
    cout << lkm;
    int a = 1;
    for (i= 0; i < lkm-i; i++)
    {
        if (mj[i] != mj[lkm - i-1])
        {
            a = 0;
            break;
        }
    }
    if (a == 1) cout << "ON PALINDROMA";
    else cout << "EI OLE PALINDROMA";
}
```

## 3.5.5 Merkkijonon kirjainten muuttaminen isoiksi

Seuraavana on algoritmi, jolla muutetaan merkkijonon kirjaimet isoiksi kirjaimiksi. Tällaista menettelyä tarvitaan hyvin yleisesti eri sovelluksissa. Kun kaikki kirjaimet ovat samaa laatua, voidaan osaltaan paremmin varmistaa merkkijonon oikeinkirjoitus ja samalla merkkijono on esimerkiksi helpompi hakea. Isoiksi kirjaimiksi muuttaminen on tärkeää esimerkiksi tietokantasovelluksissa, joissa joko lajitellaan tai haetaan tietoa jonkin merkkijonon perusteella. Erityisen tärkeää on yhdenmukainen kirjoitusasu silloin, kun ympäristö erottelee pienet ja isot kirjaimet.

### Kirjaimet isoiksi:

```
#include <iostream.h>

// Viimeistely:
// Laita vielä merkkijonon merkkien tarkistus ohjelmaan:
// Eli tarkista, että muunnettava merkki on kirjain.
// Tarkista myös, onko kirjain jo iso (ascii-koodialue)

void main()
{
    char* mj;   // sisältää merkkijonon
    int lkm, i;   /*merkkien lukumäärä merkkijonossa*/
    cout << "Anna merkkijono \n";
    cin >> mj;
    /* nyt lasketaan merkkien lukumäärä aina LOPPU-merkkiin saakka*/
    for (lkm = 0; mj[lkm] != NULL; ++lkm);
    cout << "Merkkijono on: \n" << lkm << "\n\n";
    for (i= 0; i < lkm; i++)
    {
        int a = mj[i] - 32;
         mj[i] = char(a);
    }
    cout << "merkkijono on nyt " << mj;

}
```

Esimerkiksi otsikkotiedosto ctype.h (ANSI C) sisältää funktion toupper(), joka muuntaa kirjaimen isoksi. Jos ohjelmaan ei kuitenkaan haluta sisällyttää otsikkotiedoston funktioita, voidaan hyödyntää ascii-koodia kirjaimen muuntamiseksi isoksi tai pieneksi.

Isot kirjaimet ovat ascii-taulukossa välillä 65-90 ja pienet taas kohdassa 97-122. Nähdäänkin, että erotus on 32. Näin esimerkiksi ohjelmalauseet

```
char m = 'a';
cout << m  << "\t" << char(  (int) (m-32) );
```

tulostaisivat a-kirjaimen pienenä ja isona kirjaimena.

Kaikki aakkoset (isot ja pienet kirjaimet) ja niitä vastaavat ascii-koodit saataisiin vaikkapa seuraavasti:

```
for (char m = 97; m < 123; m++)
      cout << m  << (int)m  << char( (int)(m-32) ) << (int)(m-32)  << "\n";
```

## 3.5.6 Tyhjien alkumerkkien poistaminen

Samoin perusteluin kuin edellä on merkkijonosyötöistä poistettava usein alussa olevat tyhjät merkit. Tämä tapahtuu esimerkiksi seuraavalla algoritmilla:

### Tyhjien alkumerkkien poistaminen:

```
#include <iostream.h>

void main()
{
     char mj[] = "   Tilaus";   // sisältää merkkijonon
     int lkm, paikka;   /*merkkien lukumäärä merkkijonossa*/
     cout << "merkkijono on "<< mj << "\n";
     /* nyt lasketaan merkkien lukumäärä aina LOPPU-merkkiin saakka*/
     for (lkm = 0; mj[lkm] != NULL; ++lkm);
     cout << "merkkejä on " << lkm << " kpl\n";

     // Tyhjät alkumerkit pois!
     char *temp;
     int k = 0; int s = 0;
     while (mj[k] == ' ') k++;

     for (lkm = k, s = 0; mj[lkm-k] != NULL; lkm++, s++)
     temp[s] = mj[lkm];

/*   for (s = 0; temp[s] != NULL; s++)
     mj[s] =  temp[s];  */

     cout << "merkkijono on "<< temp << "\n";
     /* nyt lasketaan merkkien lukumäärä aina LOPPU-merkkiin saakka*/
     for (lkm = 0; temp[lkm] != NULL; ++lkm);
     cout << "merkkejä on " << lkm << " kpl\n";
```

```
}
```

Tai

```
#include <iostream.h>

void main()
{
    char* mj = " Kauko";   // sisältää merkkijonon
    int lkm, i, j, k;   /*merkkien lukumäärä merkkijonossa*/
    /* nyt lasketaan merkkien lukumäärä aina LOPPU-merkkiin saakka*/
    for (lkm = 0; mj[lkm] != NULL; ++lkm);
        cout << "\nMerkkijono on siis " << mj;
    cout << "\nMerkkijonon pituus on aluksi: \n" << lkm << "\n\n";
    i = 0;
    while (mj[i] == ' ')
        i++;
        for (j = i, k = 0; j <= lkm;k++, j++)
        mj[k] = mj[j];
        mj[k] = '\0';
     for (lkm = 0; mj[lkm] != NULL; ++lkm);
    cout << "merkkijono on siis " << mj;
    cout << "\nMerkkijonon pituus on nyt:" << lkm << "\n";

}
```

Rekursiivinen algoritmi on helppo muodostaa, jos otamme käyttöön standardisuosituksen mukaisen merkin tuhoamisfunktion. String-luokka sisältää metodin remove(), jonka parametriksi annetaan tuhottavan merkin paikka.

## 3.5.7 Tyhjien loppumerkkien poistaminen

Tyhjät loppumerkit poistetaan taas vastaavasti vaikkapa seuraavasti:

### Tyhjien loppumerkkien poistaminen:

```
#include <iostream.h>

void main()
{
    char mj[] = "Tilaus     ";   // sisältää merkkijonon
    int lkm, paikka;   /*merkkien lukumäärä merkkijonossa*/
    cout << "merkkijono on "<< mj << "\n";
    /* nyt lasketaan merkkien lukumäärä aina LOPPU-merkkiin saakka*/
    for (lkm = 0; mj[lkm] != NULL; ++lkm);
    cout << "merkkejä on " << lkm << " kpl\n";

    // Tyhjät loppumerkit pois!
    char *temp;
    int k = lkm; int s = 0;

    while (mj[k-1] == ' ') {mj[k] = NULL;k--;};
// Testitulostus
    cout << "k on " << k << "\n";

    mj[k] = NULL;

// Testitulostus
    cout << "mj on " << mj << "\n";
    for (lkm = 0, s = 0; lkm <= k; lkm++, s++)
    temp[s] = mj[lkm];

/*  for (s = 0; temp[s] != NULL; s++)
    mj[s] =  temp[s];  */

    cout << "merkkijono on "<< temp << "\n";
    /* nyt lasketaan merkkien lukumäärä aina LOPPU-merkkiin saakka*/
    for (lkm = 0; temp[lkm] != NULL; ++lkm);
    cout << "merkkejä on " << lkm << " kpl\n";

}
```

Huomautettakoon, että uudessa C++-standardissa on mukana string-luokka, jossa on useita hyödyllisiä metodeja. Eräs metodeista on merkkien tuhoaminen merkkijonon alusta; tuota metodia käyttäen voidaan laatia rekursiivinen algoritmi, joka poistaa jokaisella kutsulla ensimmäisen tyhjän merkin, kunnes ensimmäisenä merkkinä on jokin muu kuin tyhjä merkki.

# 3.6 Päivämäärät

Lähes kaikissa sovelluksissa joudutaan käyttämään erityyppisiä päivämääriä. Useimmiten käyttäjää pyydetään antamaan päivämäärä jossakin tiivistetyssä, ennalta määrätyssä muodossa. Päivämäärän oikeellisuus on aina syytä tarkistaa. Seuraavassa algoritmissa päivämäärä on annettava muodossa VVKKPP, siis esimerkiksi 950911, joka on siis 11. syyskuuta vuonna 1995. Algoritmi tarkistaa, että annettu päivämäärä on 1990-luvun päivämäärä ja kuukausi ja päivä ovat oikeata suuruusluokkaa.

### Päivämäärän tarkistus:

```
#include <iostream.h>

void main()
{
char *pvm ="120988";
int vuosi, kuukausi, paiva;
do
{
paiva = (pvm[0] + pvm[1]);
kuukausi = (pvm[2] + pvm[3]);
vuosi = (pvm[4] + pvm[5]);
}
while (vuosi > 70 && vuosi < 99 && kuukausi > 0 && kuukausi < 13 &&
paiva > 0 && paiva < 32);
cout << "vuosi on " << vuosi << "\n";
cout << "kuukausi on " << kuukausi << "\n";
cout << "paivä on " << paiva << "\n";

}
```

Lisätarkistukseksi vaaditaan vielä itse syötön tarkistus virheellisten merkkien suhteen. Algoritmi ei välttämättä anna virheilmoitusta, jos jonkin numeron sijalla on kirjain. Ollaksesi täysin tarkka tulee sinun ottaa vielä huomioon kuukausien erilaiset päivien lukuäärät. Eli kuukausissa 1, 3, 5, 7, 8, 10 ja 12 on 31 päivää, kuukausisssa 4,

6, 9, 11 on 30 päivää ja helmikuussa joko 28 tai 29. 29 päivää on helmikuussa silloin, kun vuosiluku % 4 on nolla (0).

# 3.7 Parilliset ja parittomat numeroarvot

Sovellusohjelmassa on joskus tarkistettava numeroiden parillisuus. Tämä voidaan toteuttaa esimerkiksi switch .. case -lauseella:

### Numeroiden parillisuus/parittomuus:

```
#include <iostream.h>
#include <conio.h>

main()
{
int nro, Parillisia=0, Parittomia=0, Nollia = 0;

for (int k = 0; k < 10; k++)
{
cout << "Anna numero: ";
cin >> nro;
switch (nro)
{
case      2:
case    4:
case    6:
case    8:  Parillisia =Parillisia + 1;break;
case      1:
case    3:
case    5:
case    7:
case    9: Parittomia =Parittomia + 1; break;
case      0: Nollia =Nollia +1; break;
}
}
cout << "Parittomia " << Parittomia << endl;
cout << "Parillisia " << Parillisia << endl;
cout << "Nollia " << Nollia << endl;

getch();

}
```

# 3.8 SOTU-tunnusosan tarkistusmerkin muodostuminen

Henkilötunnuksen eli sosiaaliturvatunnuksen viimeinen merkki on ns. tarkistusmerkki, joka määräytyy tunnusosan yhdeksän ensimmäisen numeron perusteella. Tarkistusmerkit on esitetty alla olevassa taulukossa.

| Jakojäännös | Tarkistusmerkki | Jakojäännös | Tarkistusmerkki | Jakojäännös | Tarkistusmerkki |
|---|---|---|---|---|---|
| 0 | 0 | 11 | B | 22 | P |
| 1 | 1 | 12 | C | 23 | R |
| 2 | 2 | 13 | D | 24 | S |
| 3 | 3 | 14 | E | 25 | T |
| 4 | 4 | 15 | F | 26 | U |
| 5 | 5 | 16 | H | 27 | V |
| 6 | 6 | 17 | J | 28 | W |
| 7 | 7 | 18 | K | 29 | X |
| 8 | 8 | 19 | L | 30 | Y |
| 9 | 9 | 20 | M | | |
| 10 | A | 21 | N | | |

**Tarkistusmerkki lasketaan seuraavasti:**

SOTU-tunnuksen ensimmäiset 6 numeroa ja väliviivan jälkeiset kolme numeroa liitetään yhteen.
Esimerkiksi tunnuksen 110977-123S liitetty numero-osa olisi 110977123.

Seuraavaksi tuo saatu luku jaetaan luvulla 31 ja tarkastellaan jakojäännöstä.
Esimerkiksi edellä saadun luvun 110977123 ja luvun 31 jakojäännös on 6.

Tuota jakojäännöstä käytetään indeksi tarkistusmerkkitaulukkoon.
Esimerkissämme saatua lukua 6 vastaava merkki on taulukon mukaan 6.

Esimerkkimme SOTU on siis virheellinen; sen pitäisi olla 110977-1236.

Seuraavana on ohjelma, jolla tarkistetaan sotu-tunnuksen tarkistusmerkin oikeellisuus:

**Sotu-merkki:**

```
#include <iostream.h>
#include <stdlib.h>


void main()
{

    char sotu[] = "040368-012C";
    char tarkmerk[] = { '0', '1', '2', '3', '4', '5', '6', '7',
'8', '9', 'A', 'B', 'C', 'D', 'E', 'F', 'H', 'J', 'K', 'L',
'M', 'N', 'P', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Z'};
    char *numerot;

    char m;      // etsittävä merkki
    int i, j;    // paikka merkkijonossa //
    int on;

    m = sotu[10];  // Annettu sotu-merkki
    for (i = 0; i < 6 ; i++)
    numerot[i] = sotu[i];
    for (i = 7, j = 6; i < 10 ; i++, j++)
        numerot[j] = sotu[i];


    long lukuna = atol(numerot);  // Jos int 2 tavua
    cout << lukuna << "\n";
    int jakoj = lukuna % 31;

    if (tarkmerk[jakoj] == m) cout << "Oikein! \n";
    else cout << "Virheellinen sotu";

}
```

# 3.9 Merkki numeroksi

Merkkimuotoinen tieto voidaan muuttaa numeroarvoksi muuntamalla char-tyyppi int-tyypiksi. Tällöin merkki kuitenkin esitetään merkkiä vastaavana ascii-koodina. Tuosta ascii-koodista tulee vähentää merkkiä 0 (merkkinä ‘0’) vastaava ascii-koodi eli luku 48.

Seuraavana on esimerkkiohjelma:

**Merkki numeroksi:**

```
#include <assert.h>
#include <ctype.h>
void  main()
{
 int lukuna;
 char nro;
 cout << "Anna numeroarvo väliltä 0 -9 " << endl;
 cin >> nro;
 assert(isdigit(nro) );
 lukuna = nro - 48;
 cout << lukuna << endl;
 cout << ++lukuna << endl;

 getch();
}
```

# 3.10 Merkkien ja merkkijonojen käsittely tiedostoissa

Tässä jaksossa käsitellään hieman syöttö- ja tulostusvirtoja sekä tiedostojen käsittelyä.

## Tiedostoihin liittyvät virrat

**Virta** on peräkkäisiä tavuja.
Virrat käsitellään aina yhteneväisellä tavalla.
Pääluokka on **ios.**

**ios-aliluokat**: **istream** ja **ostream**.
istream hoitaa luettavat virrat
ostream hoitaa kirjoitettavat virrat
Otsikkotiedosto on iostream.h.

Luokilla **ifstream ja ofstream** kirjoitetaan ja luetaan tiedostoja.
Otsikkotiedosto on **fstream.h.**

**Luokat, joilla voidaan kytkeä virrat päämuistiin:**
Luokkia **istrstream ja ostrstream** käytetään käsiteltäessä merkkikenttiä virtoina.
Otsikkotiedosto on **strstream.h**

Luokkia **istringstream** ja **ostringstream** käytetään haluttaessa kytkeä virtoja string-luokan olioihin.
Otsikkotiedosto on **sstream.h**

**Esimääriteltyjä virtaolioita:**
**cin, cout, cerr ja clog**.

cin kuuluu istream-luokkaan
cout, cerr ja clog kuuluvat ostream-luokkaan
cerr tulostaa virheilmoituksia
clog hoitaa kirjoitukset ohjelmalokeihin

Luokassa **ios** on lippuja, joilla annetaan virtojen tila. Virta on OK, jos mikään lippu ei ole päällä.

| Lippu | Merkitsee |
|---|---|
| Failbit | Luku/kirjoitus epäonnistui |
| Eofbit | Tiedoston loppu eteen viime luvussa |
| Badbit | Paha virhe sisäisessä puskurissa |

**ios** sisältää useita metodeita:

| Funktio | Merkitys |
|---|---|
| void clear() | Tilaliput nollataan |
| bool good() | Antaa true, jos liput ei päällä |
| bool fail() | Antaa true, jos failbit tai badbit päällä |
| bool eof() | Antaa true, jos eofbit päällä |
| bool bad() | Antaa true, jos badbit päällä |
| bool operator!() | Antaa tuloksen fail() |
| operator bool() | Antaa tuloksen not fail() |

## Esimerkki, jossa cin-olion metodeita kokeillaan.

```
if (cin.fail())
{
    cerr << "Luku epäonnistui ";
    cin.clear();
}
```

Lause

```
if (cin.fail())
```

voidaan kirjoittaa myös

```
if (!cin)    // koskapa taulukossa on operator!
```

Esimerkiksi lippu eofbit menee päälle, jos yritetään lukea tiedoston loputtua ja tällöin metodi eof() antaa tuloksen true.

## Virtojen lukeminen

### Muotoiltu ja muotoilematon luku

Muotoilematon luku lukee tietoa syöttövirrasta juuri kuten se on virrassa.
Muotoiltu luku voi tulkata tietoa.
Luku päättyy aina seuraavaan tyhjään merkkiin.

## Tiedostot

Luokat: **ifstream** ja **ofstream**

### Esimerkki: Lasketaan tiedoston rivi- ja merkkimäärä.

```
#include <iostream.h>
#include <fstream.h>

void  main()
{
    int n=0, m=0;
    char c;
    ifstream f1("apu.txt");  // f1 on luokan ifstream virta,se
avataan
    while (f1.get(c))
    {
      if (c != '\n') m++;
      if (c == '\n')
         n++;
    }
    cout << " Tiedosto koostuu " << n<< " rivistä" << endl;
cout << " Tiedosto koostuu " << m<< " merkistä" << endl;
```

```
}
```

Lauseen

```
ifstream f1("apu.txt");
```

sijaan voidaan kirjoittaa

```
ifstream f1;
f1.open("my_file.txt");
```

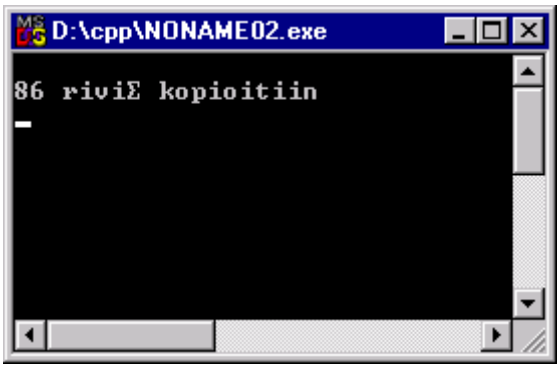
Jos tiedosto on muualla, täytyy antaa polku:

(esim. c:\\temp\\apu.txt)

Huomaa, että kenoviiva (\) on varattu symboli. Jos tiedoston nimi olisi esimerkiksi tabu.txt, syntyisi yhdellä kenoviivalla merkintä \tabu.txt, jolloin syntyisi sarkainsiirto (\t).

## Esimerkki: kopioidaan tiedosto ja lasketaan rivit

```
#include <iostream.h>
#include <fstream.h>
void  main()
 {
   int n=0;
    char c;
    ifstream f1("e:\\cpp\\fil1\\apu.txt");
    ofstream f2("e:\\cpp\\fil1\\apu_var.txt");
    while (f1.get(c))
   {
      f2.put(c);
           if (c == '\n')
               n++;
    }
   cout << endl << n << " riviä kopioitiin" <<endl;
 }
```




Kun tiedosto avataan, voidaan antaa käsittelyominaisuudet:

```
ifstream f(tiedostonimi, moodi);
```

moodi on muotoa

```
ios::lippu | ios::lippu | jne
```

| Lippu | Moodi |
|---|---|
| in | Tiedostoa luetaan. Sen on oltava olemassa. |
| out | Tiedostoon kirjoitetaan. Jos sitä ei ole, se luodaan. Jos tiedosto on, sen päälle kirjoitetaan. |
| app | Tiedostoon kirjoitetaan. Jos sitä ei ole, se luodaan. Jos tiedosto on, sen loppuun kirjoitetaan. |
| trunc | Tiedoston päälle kirjoitetaan. |
| ate | Mennään heti avauksessa tiedoston loppuun. |
| binary | Tiedostoa käsitellään binääritiedostona. |

Esimerkki

```
f.open("kk.txt", ios::in | ios::out);
```

Tiedostoa päivitetään (luku ja kirjoitus). Tiedoston on oltava olemassa.

**Tiedostopuskuri.**

**Tiedostopuskuri**
Puskuriin luettava ja kirjoitettava tieto.

Puskurin käsittely: rdbuf().
rdbuf() palauttaa osoittimen puskuriin.

Esimerkiksi:
Tuloste koko tiedostosta f1 näytölle:

```
cout << f1.rdbuf();
```

tiedosto f1 kopioidaan tiedostoon f2:

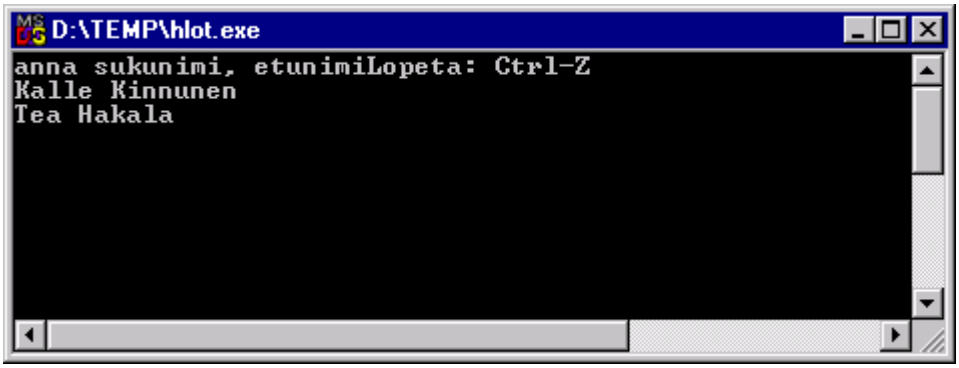
```
f2 << f1.rdbuf();
```

### Esimerkki: tiedostoon syötetään tietoa ja tulostetaan tiedosto:

```
#include <iostream.h>
 #include <fstream.h>
 #include <iomanip.h>
main()
 {
    ofstream fout("d:\\temp\\hlot.txt", ios::app);
    char etunimi[20], sukunimi[30];
    cout << "anna sukunimi, etunimi" << "Lopeta: Ctrl-Z" << endl;
    while (cin>>setw(20)>>etunimi && cin>>setw(30)>>sukunimi)
    fout << etunimi << ' ' << sukunimi << endl;
    fout.close();
    // Tulostetaan tiedosto
    ifstream fi("d:\\temp\\hlot.txt", ios::in);
    cout << "Henkilötiedot:" << endl;
    cout << fi.rdbuf();
 }
```

Tiedosto apu.txt aluksi:


Ajetaan ohjelma ja lisätään pari riviä tietoja:

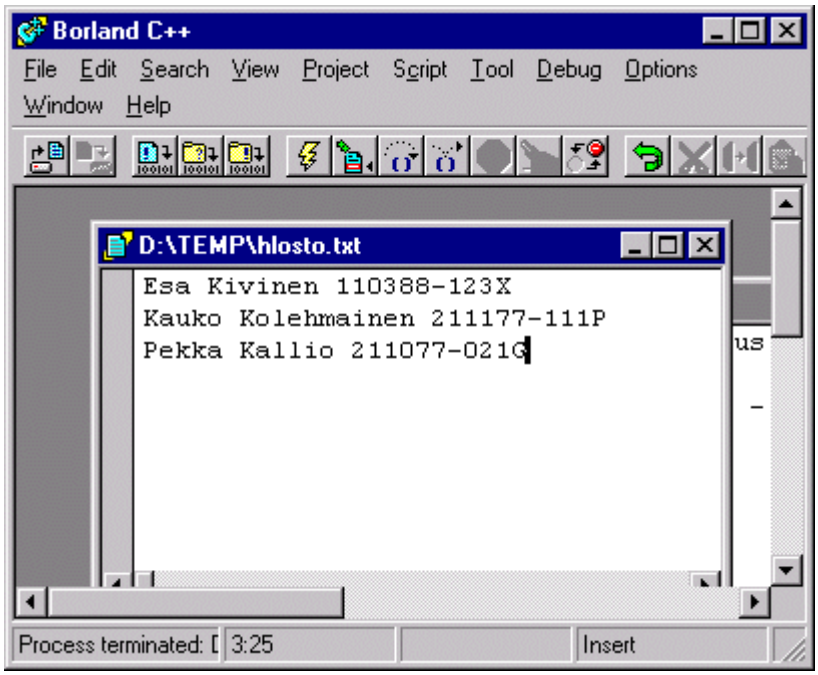


Tiedosto apu.txt lisäysten jälkeen:


## Esimerkki: Henkilöstön tiedot luetaan tiedostosta hlosto.txt ja tarkistetaan, onko henkilöllä tänään syntymäpäivä

Tiedosto sisältää henkilön etu- ja sukunimen sekä sotu-tunnuksen. Sotu-tunnuksen kautta selvitetään, onko jollakulla henkilöllä tänään syntymäpäivä. Jos tällainen henkilö löytyy, sijoitetaan henkilön nimi ja ikä erilliseen tiedostoon ja tulostetaan myös näytölle.

hlosto.txt




Nykyisen päivämäärän selvittämiseen on nyt käytetty otsikkotiedoston dos.h getdate()-funktiota, joka ottaa parametrikseen date-tyyppiä olevan tietuemuuttujan. Date-tietue sisältää kentät, joiden kautta saadaan esille päivä ja kuukausi. Tiedot otetaan tiedostosta ensin 2-ulotteiseen taulukkoon. Taulukon ensimmäinen sarake kuvaa rivinumeroa ja toinen tekstiriviä. Kun tiedetään, että sotutunnus on rivin viimeisenä merkkijonona, voidaan sotutunnuksesta siepata henkilön syntymäpäivä ja –kuukausi, jota verrataan sitten nykyiseen päivään ja kuukauteen.

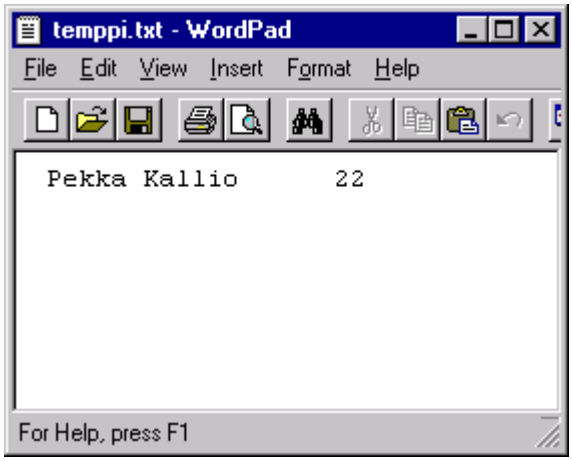
## Syntymäpäivän selvittäminen henkilötiedoston ja sotutunnuksen kautta

```
#include <iostream.h>
#include <fstream.h>
#include <iomanip.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <dos.h>
#include <conio.h>

#define and &&
#define or ||

 main()
{
struct date d;
getdate(&d);
```

```
int pp = d.da_day;
int kk = d.da_mon;

char nimi[20];
char paiva[3];
char kuukausi[3];
char* vuosi = "    ";
int ika;
char rivi[5][80];
int n= 0, w = 0;  char c;
    bool loytyi = false;
      ifstream fin("hlosto.txt");
      ofstream fut("temppi.txt");
     while (fin.get(c))
  {
      if (c == '\n') {rivi[w][++n] = '\0'; w++; n=0; }
         else rivi[w][n++] = c;
  }
      int k = 0, s;

      for (k=0; k < w; k++)
      {
      for (s=0; rivi[k][s] != '\0'; s++); // lasketaan kunkin rivin
pituus
      int pituus = s;

         vuosi[0] = '1'; vuosi[1]= '9';
      vuosi[3] = rivi[k][pituus-7];
      vuosi[2] = rivi[k][pituus-8];
      kuukausi[0] = rivi[k][pituus-10];kuukausi[1] = rivi[k][pituus-
9];
      kuukausi[2] = '\0';
      paiva[0] = rivi[k][pituus-12]; paiva[1] = rivi[k][pituus-11];
      paiva[2]= '\0';
         cout << atoi(vuosi) << endl;
        if ( (atoi(paiva) == pp) and (atoi(kuukausi) == kk ) )
          {
         int vv= d.da_year;
         loytyi = true;
         strncpy(nimi, rivi[k], pituus -12);
         nimi[pituus-12] = '\0';
            fut << nimi  << '\t'<<  (vv - atoi(vuosi))  << endl;
         ika = vv - atoi(vuosi);
         }
       }
    if (loytyi)
   {
     cout << nimi << endl;
     cout << ika << endl;
    }
   getch();
```

```
}
```

Ohjelman ajon jälkeen näyttää temppi.txt seuraavalta:




## Sanomien salaaminen

Seuraava ohjelma salaa sanomia. Ennen salaamisen aloittamista kysytään sanomaa ja salausavainta. Salausavain on esimerkiksi numeroarvo, jonka mukaan haetaan alkuperäisen kirjaimen paikalla avainarvon verran oikealla oleva kirjain (esimerkiksi, jos avain on 1, vastaa kirjainta A kirjain B jne). Ohjelma kirjoittaa sanoman salattuna tiedostoon. Sitten ohjelmassa haetaan tiedosto ja puretaan se selväkieliseksi ja tulostetaan.

### Salattu sanoma tiedostoon sekä sen purkaminen:

```
#include <iostream.h>
#include <fstream.h>
#include <iomanip.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <dos.h>
#include <cstring.h>
#include <conio.h>
#define and &&
#define or ||

 main()
{
char * sanoma;
int avain;
char * s = "";
```

```
cout << "anna sanoma: " << endl;
cin >> s;
cout << "Anna avain: " << endl;
cin >> avain;
int k;
for (k =0; s[k] != '\0'; k++);
char * temp = new char[k];
for (k =0; s[k] != '\0'; k++)
{   temp[k] = (char) (int(s[k]) + avain); }
temp[k] = '\0';

   cout << "Viesti salattuna \n";
   cout << temp << endl;

  ofstream fout("salattu.txt");
           fout << temp << endl;

   cout << "Haetaan salattu viesti ja puretaan \n";
   ifstream f("salattu.txt");
    char * haettuviesti = new char[k];
   f >> haettuviesti;
   int p;
    for (p=0; haettuviesti[p] != '\0'; p++);

    char * selva = new char[p];
    for (int p =0; haettuviesti[p] != '\0'; p++)
    {
   selva[p] = (char) (int(haettuviesti[p]) - avain);
   }
   selva[p] = '\0';
    cout << "Tulostetaan selvänä\n";
   cout << selva << endl;

   getch();
    }
```

Jos haluat siepata erillisiä sanoja (eli välilyöntejä on mukana), voidaan sanoma ottaa getline-funktiolla (cin.getline(sanoma, puskurikoko)). Getline sieppaa myös rivinvaihdon.