Tieteellinen laskenta II
Loppuprojekti s2015

Datan piilottaminen kuvatiedostoon

Mikko Kangasmäki

1. Johdanto

Tehtävänä on piilottaa dataa .ppm-kuvatiedostoon.

Toteutuksessani tiedostoon voi piilottaa mitä tahansa dataa, mutta teksti on erityisasemassa ja sen voi piilottaa ja lukea suoraan komentorivillä.

2. Metodit

Ppm-tiedostossa kuva on jaettu yksittäisiin pikseleihin, joista jokaisesta on tallennettu kolme lukua: pikselin punaisuus, sinisyys ja vihreys (r,g,b). Tyypillisesti yhden rgb-värin sävy ilmoitetaan luvulla 0-255, mutta standardi sallii myös muut luvut, suurimmillaan 65 536.[1] Yhden pikselin yhden värin sävy siis mahtuu yhteen tai kahteen tavuun.

Ohjelmani koodaa piilotettavan datan näihin pikselitietoihin siten, että jokaisesta värisävyn ilmoittavasta luvusta muutetaan vähiten merkitsevää bittiä.

Jos kuvassa esimerkiksi on 255 eri rgb-sävyä, koodattaisiin kirjain “a” dataan näin:

a = 97 asciina = 01100001 =

alkup. data	a	koodattu
1110000**1**	**1**	1110000**1**
1000011**1**	**0**	1000011**0**
0110011**1**	**0**	0110011**0**
0101011**1**	**0**	0101011**0**
0000000**0**	**0**	0000000**0**
0111001**0**	**1**	0111001**1**
0000100**0**	**1**	0000100**1**
0110101**1**	**0**	0110101**0**

Jos rgb-sävyjä olisi enemmän, a koodattaisiin vain jokatoiseen tavuun.

1 http://netpbm.sourceforge.net/doc/ppm.html

3. Toteutus

Ohjelma jakautuu kuuteen komponenttiin:

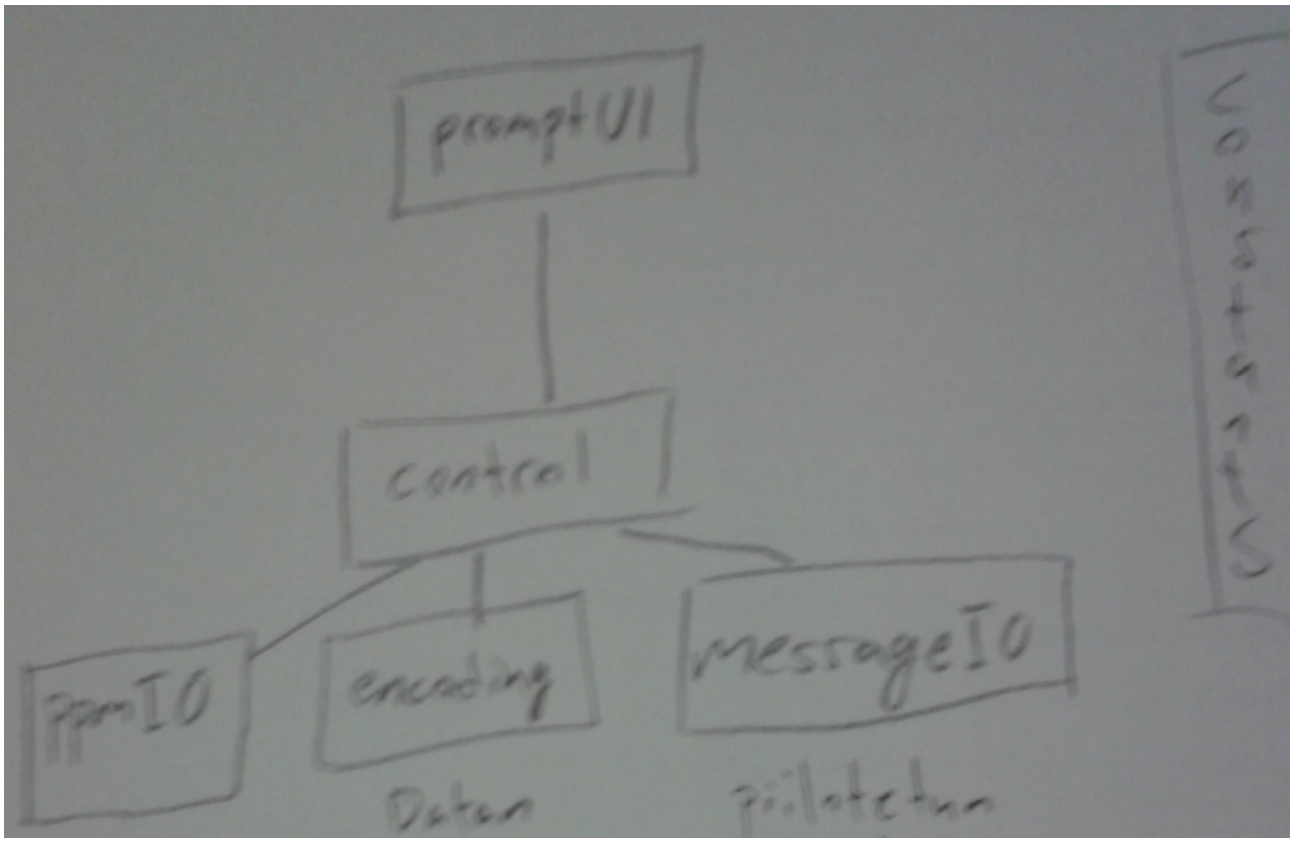
`constants.f90`

`promptUI.f90`

`control.f90`

ja

`ppmIO.f90`
`encoding.f90`
`messageIO.f90`



Modulissa `constants` on muiden modulien yhteiset vakiot.
Moduli `promptUI` on käyttöliittymä terminaalille.
Moduli `control` hoitaa kaikki isommat tehtävät, joita ohjelma tekee: piilottaa datan kuviin tai dekoodaa datan kuvasta.

Modulit `ppmIO, encoding ja messageIO` hoitavat kaikki pienemmät tehtävät:
`ppmIO` lukee ja kirjoittaa .ppm-tiedostoja.
`encoding` ottaa vastaan raakadataa ja koodaa siihen viestin tai dekoodaa siitä viestin ulos.
`messageIO` lukee ja kirjoittta formatoimatonta dataa levylle ja muuntaa tekstiä ascii-koodiksi.

Modulien tarkempi toiminta on dokumentoitu f90-tiedostoissa itsessään. Alimman tason modulien oleellisimmat toiminnot ovat seuraavat:

`ppmIO:`
`read_metadata`: Lukee metadatan kuvatiedostosta (esim kuvan korkeus, leveys ja värien määrä).
`read_data:` Lukee pikselidatan kuvatiedostosta.
`write_data:` Kirjoittaa meta- ja pikselidatan kuvatiedostoksi

`encoding:`
`encode:` Piilottaa pikselidataan viestin
`decode:` Etsii pikselidatasta viestin

messageIO:
`read_raw_data:` Lukee tiedoston sisällön välittämättä sen formaatista.
`write_raw_data:` Kirjoittaa tiedostoon raakadataa ilman formaattia.
`text_to_ascii:` Muuttaa tekstin ascii-numeroiksi
`ascii_to_text:` Muuttaa ascii-numerot tekstiksi.

Piilotttaessaan dataa ohjelma toimii näin:
`promptUI` saa käyttäjltä käskyn piilottaa dataa ja pyytää `control`ia toteuttamaan käskyn
`control` pyytää `ppmIO`:lta kuvan metadatan
`control` pyytää `ppmIO`:lta kuvan pikselidatan
`control` pyytää `messageIO`:lta piilotettavaa dataa
`control` pyytää `encoding`ia piilottamaan datan kuvan pikselidataan
`control` pyytää `ppmIO`:a kirjoittamaan kuvan piilotetun datan kanssa uudeksi tiedostoksi.

Lukiessaan dataa ohjelma taas tekee näin:
promptUI saa käyttäjältä käskun lukea data kuvasta ja pyytää controlia toteuttamaan käskyn
`control` pyytää `ppmIO`:lta kuvan metadatan
`control` pyytää `ppmIO`:lta kuvan pikselidatan
`control` pyytää `encoding`ia lukemaan piilotetun datan pikselidatasta
`control` pyytää `messageIO`:a kirjoittamaan piilotetun datan tiedostoon.

Ohjelman toimintaa on turha eritellä tarkemmin tässä. Koodiin on kirjoitettu melko kattavat seloteot yksittäisten alirutiinien toiminnasta. Käytetyt algoritmit ovat hyvin suoraviivaisia. Esimerkiksi merkkien lukeminen kunnes vastaan tulee whitespace.

Datan piilottamisessa käytettyjä bitshift-operaatioita ei käsitelty kurssilla, mutta nekin ovat aika simppeleitä. Lähinnä päänvaivaa tuotti oikeiden indeksien löytäminen. Etenkin, kun jotkut bittitason komennut aloittavat indeksoinnin nollasta.[2]

2 Minun varmaan pitäisi kirjoittaa niistä enemmän todistaakseni, että olen ymmärtänyt ne, mutta uskon ja toivon, että koodini todistaa sen puolestani. Lisäksi en enää ehdi.

4. Käyttö

Yksityiskohtaisemmat käyttöohjeet löytyvät src-kansio tiedostosta README.txt. Tässä oleellisimmat:

Käännä: kirjoita terminaaliin

```
gfortran *.f90
```

kaksi kertaa. Ensimmäisellä kerralla gfortran tekee vain mod-tiedostot.

Koodaa tekstiä kuvaan:

```
./a.out kuva.ppm kohde.ppm text="Salainen viesti"
```

Salainen viesti tulee olemaan tiedostossa `kohde.ppm`, `kuva.ppm` ei muutu.

Koodaa tiedosto kuvaan:

```
./a.out kuva.ppm kohde.ppm tiedosto.xxx
```

`tiedosto.xxx` koodataan raakamuodossa kuvaan. Jälleen `kuva.ppm` ei muutu ja `tiedosto.xxx` on piilotettu `kohde.ppm`:ään.

Tulosta salattu viesti terminaaliin:

```
./a.out kuva.ppm
```

olettaen, että tiedostossa `kuva.ppm` on salattu tekstiviesti.

Tallenna salattu tiedosto:

```
./a.out kuva.ppm tiedosto.xxx
```

tallentaa tiedostoon `kuva.ppm` salatun datan tiedostoon `tiedosto.xxx`.
Ohjelma ei osaa automaattisesti tarkistaa datan formaattia.

Quick start:

Kansiossa run on Koodattu Kekkosen kuvaan Kalevalan kolme ensimmäistä runoa. Tulosta ne ruudulle

```
./a.out kalevala_kekkonen.ppm
```

`tai tiedostoon`

```
./a.out kalevala_kekkonen.ppm kalevala.txt
```

Samassa kansiossa olevaan tiedostoon `ukk2.ppm` on piilotettu jpg-kuva[3], jonka saa tiedostoon komennolla

```
./a.out ukk2.ppm hidden.jpg
```

3 Kuva on sivulta http://people.sc.fsu.edu/~jburkardt/

5. Toiminta käytännössä

Ohjelma toimii moitteettomasti ja nopeasti. Tosin, olen kääntänyt sen vain gfortranilla Ubuntu 14.04:ssa, enkä ole ehtinyt testata sitä kuviin, joissa värejä on eri määrä kuin 255. Koodissa on myös piilossa joitakin mahdollisia ongelmakohtia, joihin en enää ehtinyt puuttua. Niistä enemmän seuraavaksi:

6. Mitä olisi voinut tai pitänyt tehdä toisin, jos olisi aikaa ja projekti olisi tärkeämpi

Salatun tiedoston koko on koodattu integer-muuttujaksi. Ohjelma kyllä huomioi mahdolliset eri integer-muuttujien koot, mutta olisi ollut parempi käyttää vakiokokoista muuttujaa, esimerkiksi integer*4:sta. Tämä voisi vaatia muutoksia moneen kohtaan koodia, jos kone ei osaa lukea integer*4:ia integereiksi oikealla tavalla.

Modulissa `ppmIO` on osittain redundanttia koodia. Se voisi käyttää apuna `messageIO`:n alirutiinia, joka lukee ja kirjoittaa raakadataa.

Ohjelma voisi tunnistaa kommenttirivin kuvasta. Tätä ei olisi edes vaikea toteuttaa: etsi ensin # ja sen jälkeen seuraava rivinvaihto.

Ohjelma ajattelee, että .ppm-kuvan metadatassa on vain yksi whitespace värien määrän jälkeen. Tämä on todellisuudessa standardin mukaista, ainakin jos sivua http://netpbm.sourceforge.net/doc/ppm.html on uskominen. Tehtävänanto antaa kuvan, että whitespaceja voi olla useampikin. Olisi ollut helppoa (mutta tylsää) toteuttaa ohjelma niin, että se laskee jäljellä olevien pikseleiden määrästä, mistä pikselidata alkaa. Pelkkien whitepacien ohikelaaminen ei tässä auta, sillä ensimmäinen pikselidata voi olla jonkin whitespacen ascii-koodi.

Ohjelma voisi tunnistaa automaattisesti, millainen tiedosto kuvaan on piilotettu. Tämän voisi toteuttaa piilottamalla tiedostopääte kuvan alkuun, jolloin ohjelma voisi automaattisesti kirjoittaakin piilotetun datan oikeaan muotoon.

Help-teksti on kovakoodattu `control`-moduliin. En tiedä, miten tällaisia asioita yleensä hoidetaan. Olisi vähän ikävää ja kömpelöä kuljettaa helppiä tekstitiedostona koodin mukana. Toisaalta, ongelmana on myös se, että tämä vähentää ohjelman modulaarisuutta. Parempi ratkaisu olisi ehkä, että `control` välittää help-tekstin käyttöliittymälle, joka sitten päättää, miten sen tulostaa. Samoin, virhetilanteissa jokin ei-käyttöliittymäluokka saattaa tulostella ruudulle. En tosin tiedä, onko fortranin kanssa edes järkevää ajatella käyttöliittymiä.