Kurssin TIEA211 Algoritmit 2 ohjelmointitehtävät

Antti Valmari

7. kesäkuuta 2023

1 Yleistä

Kunkin ohjelman saa tehdä millä tahansa TIMin tukemalla ohjelmointikielellä. Ohjelma palautetaan TIMissä sijaitsevalle automaatille, joka testaa sitä monipuolisesti erilaisilla syötteillä. Jos automaatti ei havaitse virheitä, se kirjaa opiskelijalle pisteen merkiksi siitä, että palautus on hyväksytty. Muussa tapauksessa automaatti palauttaa testisyötteen jolla virhe ilmeni sekä kuvauksen virheestä.

Kukin ohjelma lukee syötteensä standardi-inputista ja tuottaa tulosteensa standardi-outputtiin. Ohjelma saa luottaa siihen, että syöte noudattaa tehtävän kuvauksessa ilmoitettua syntaksia. Ohjelman ei siis tarvitse tunnistaa ja raportoida virheellistä syötettä. Esimerkiksi jos syntaksi lupaa, että seuraavaksi tulee kokonaisluku, niin voit huoleti käyttää ohjelmointikielten valmista toimintoa, joka ohittaa välilyönnit ja rivinsiirrot sekä lukee kokonaisluvun. Kuitenkin, jos haluat, niin saat laittaa ohjelmasi ilmoittamaan lupausten vastaisesta syötteestä. Liitteessä Liite A on kerrottu, miten syötettä voi lukea Javalla, C++:lla ja C#:lla.

Testausteknisistä syistä[1] joidenkin ohjelmien tulee kyetä käsittelemään yhden ajon aikana monta erillistä tehtävää. (Siitä on hyötyä myös oppimisen kannalta: tulee harjoiteltua muuttujien uudelleen alustamista.) Siitä selviää yleensä näppärästi laittamalla pääohjelmaan silmukan, joka toistaa yhden tehtävän käsittelyä kunnes saa tiedon, että syötettä ei enää ole jäljellä. Useimmat opettajan tekemät ohjelmointitehtävien ratkaisut noudattavat jompaakumpaa seuraavista rakenteista:

```
while( yksi_tehtava() );
while( hae_tiedot() ){ ratkaise_tehtava(); }
```

Jotta ohjelman tuottaman vastauksen voisi testata vertaamalla merkki merkiltä mallivastaukseen, on osan tehtävistä tuottama tuloste määritelty pikkutarkemmin kuin pelkästään ohjelman kuvitellun käyttötarkoituksen kannalta olisi tarpeen.

Ohjelmointitehtävien opetuksellisten tavoitteiden kannalta epäolennaisia asioita on jätetty testausympäristöissä vähälle huomiolle. Testausympäristöt eivät testaa ohjelmia täydellisesti edes olennaisten asioiden osalta. Täydellinen testaaminen ei ole mahdollista teoriassa eikä käytännössä. Siksihän algoritmejakin todistetaan oikeiksi, että niiden toimivuutta ei voi tarkastaa kattavasti pelkästään testaamalla.

[1] Ohjelman käynnistäminen vie testaukseen käytettävässä ympäristössä suhteettomasti aikaa verrattuna ohjelman ajamiseen pienellä syötteellä. Siksi haluttiin, että samalla käynnistämisellä voi ajaa monta pientä testitapausta. Jos ohjelma tekee jollain niistä virheen, testausympäristö ajaa ohjelman uudelleen pelkästään sillä testitapauksella, jolla ensimmäinen virhe ilmeni. Jos ohjelma tekee nytkin virheen, niin virheilmoitus annetaan pelkästään tätä testitapausta koskien. Muussa tapauksessa virheilmoitus annetaan koko ensimmäisen ajon syötettä koskien.

2 Tehtävä: Perinnönjako

Ohjelman tehtävänä on jakaa perinnön arvo perillisille. Jos perittävällä on jälkeläisiä elossa, perintö jaetaan heille. Jokainen lapsi joka elää tai jolla on elossa olevia jälkeläisiä saa yhtä suuren osan. Kuolleen lapsen osa jaetaan samalla tavalla hänen jälkeläisilleen. Jos perittävällä ei ole jälkeläisiä elossa, niin palautuksessa 1 perintö annetaan valtiolle ja palautuksissa 2 ja 3 perintö jaetaan samalla tavalla hänen vanhemmilleen. Jos silläkään tavalla ei löydy elävää perillistä, perintö annetaan valtiolle. (Ohjelmassa ei siis oteta huomioon sitä, että Suomen lain mukaan perintö menee joissakin tapauksissa puolisolle tai isovanhemmille, sedille ja tädeille.)

Perintö esitetään kokonaislukuna. Jos jako lapsille tai vanhemmille ei mene tasan, niin jakojäännös jätetään jakamatta perinnöksi. Tämän säännön tärkein tehtävä on varmistaa, että pyöristysvirheet eivät häiritse opiskelijoiden ohjelmien vastausten vertaamista malliohjelman vastauksiin. Se myös varmistaa, että ohjelmat eivät missään tilanteessa jaa perinnöiksi hitustakaan enempää kuin mitä on jaettavaa. On parempi, että ohjelma jättää hieman jakamatta, kuin että perintörahoja maksettaessa viimeisen saajan kohdalla huomataan, että hänelle ei voida antaa ihan niin paljoa kuin luvattiin, koska raha loppui kesken.

Henkilöt on numeroitu ykkösestä alkaen peräkkäisillä kokonaisluvuilla siten, että lapsen numero on aina suurempi kuin hänen vanhempiensa numerot.

Syötteessä on nolla tai useampia perinnönjakotehtäviä ja lopuksi 0. Perinnönjakotehtävässä on tässä järjestyksessä perittävän numero, perittävä summa, henkilöiden tiedot ja 0. Henkilön tiedoissa on hänen numeronsa (jos hän elää) tai sen vastaluku (jos hän on kuollut), hänen nimensä merkkijonona jossa ei ole välilyöntejä eikä ympärillä lainausmerkkejä, yhden vanhemman numero tai 0, ja toisen vanhemman numero tai 0. Henkilöiden tiedot annetaan kasvavassa numerojärjestyksessä. Tekstialkioiden välissä voi olla vapaasti välilyöntejä ja rivinsiirtoja.

Esimerkki:

```
4 7825349
-1 Klaara 0 0 -2 Vihtori 0 0
 3 Amalia 0 2
-4 Bernard 2 1
 5 Cecilia 1 2
-6
Erika    0

    0
7 Ferdinand 6 0
8 Daniel 0 1 0 1 9 -1 ToinenVainaja 0 0 0 0
2 syöte_loppui_ennen_tätä_kohtaa 1 0 0 0
```

Ohjelma saa luottaa siihen, että syöte noudattaa syntaksia ja kaikki siinä annettavat tiedot ovat teknisesti mielekkäitä. Ohjelma saa luottaa esimerkiksi siihen, että syötteessä annettava vanhemman numero on vähintään 1 ja pienempi kuin henkilön oma numero. Ohjelma ei saa luottaa siihen, että henkilöiden tai sukupolvien määrä on niin pieni kuin se olisi todellisessa perinnönjaossa. Palautuksissa 1 ja 2 ohjelma saa silti luottaa siihen, että sukupolvia on vähän (alle 10), mutta palautuksessa 3 sukupolvia voi olla paljon.

Ohjelman tulee tulostaa numerojärjestyksessä jokaisesta, joka saa enemmän kuin 0, rivi, jossa on aluksi henkilön nimi tai ”Valtio”, sitten ” saa ”, sitten summa ja lopuksi rivinsiirto. Niiden perään ohjelman tulee tulostaa rivi ”Jakamatta jää ”, summa ja rivinsiirto. Muuta ohjelma ei saa tulostaa. Kuitenkin, jos syöte ei noudata edellä annettuja lupauksia, ohjelma saa (mutta sen ei tarvitse) tulostaa opiskelijan valitseman virheilmoituksen. Edellä annetusta esimerkkisyötteestä ohjelman pitää tulostaa:

```
Amalia saa 1956337
Cecilia saa 3912674
Daniel saa 1956337
Jakamatta jää 1
Valtio saa 9
Jakamatta jää 0
```

Testiaineistot on siten laadittu, että jokainen palautukseksi 2 kelpaava ohjelma kelpaa myös palautukseksi 1, ja jokainen palautukseksi 3 kelpaava ohjelma kelpaa myös palautukseksi 2.

Tässä tehtävässä ei tarvita erityisiä algoritmeja ja tietorakenteita, vaan perustason ohjelmointikeinot riittävät. Vaiheita 2 ja 3 vaikeuttaa se, että henkilö saattaa saada perintöä sekä toiselta että toiselta vanhemmaltaan. Siksi perintösumma kannattaa jakaa käymällä henkilöt läpi numerojärjestyksessä siten, että kunkin henkilön kohdalla joko tulostetaan hänen saamansa summa tai jaetaan se hänen niille lapsilleen, joille se kuuluu jakaa. Jakamisen lapsille voi vaihtoehtoisesti toteuttaa niinpäin, että lapsen kohdalla haetaan hänelle tuleva perintösumma hänen vanhempiensa tiedoista. Näillä tavoilla jokainen henkilö on saanut hänelle tulevan perinnön kokonaan ennen kuin se tulostetaan tai jaetaan hänen lapsilleen.

Opettajan ratkaisu on jonkin verran yli 100 riviä pitkä.

3 Tehtävä: Lukujärjestys

Tähän tehtävään tutustuminen on ehkä helpointa aloittaa esimerkin avulla. Keväällä 2023 kurssilla TIEA211 oli seuraavat tapahtumat:

```
TIEA211 Algoritmit 2 Kevät 2023
21.03.2023 10-12 B103 Luento
23.03.2023 14-16 B103 Luento
23.03.2023 16-18 C231.1 Neuvonta
27.03.2023 10-12 B122.1 Neuvonta
28.03.2023 10-12 B103 Luento
30.03.2023 14-16 B103 Luento
30.03.2023 16-18 C231.1 Neuvonta
11.04.2023 10-12 B103 Luento
13.04.2023 14-16 B103 Luento
13.04.2023 16-18 C231.1 Neuvonta
17.04.2023 10-12 B122.1 Neuvonta
18.04.2023 10-12 B103 Luento
20.04.2023 14-16 B103 Luento
20.04.2023 16-18 C231.1 Neuvonta
24.04.2023 10-12 B122.1 Neuvonta
25.04.2023 10-12 B103 Luento
27.04.2023 14-16 A102 Luento
27.04.2023 16-18 C231.1 Neuvonta
02.05.2023 10-12 B103 Luento
04.05.2023 14-16 B103 Luento
04.05.2023 16-18 C231.1 Neuvonta
08.05.2023 10-12 B122.1 Neuvonta
09.05.2023 10-12 B103 Luento
11.05.2023 14-16 B103 Luento
11.05.2023 16-18 C231.1 Neuvonta
15.05.2023 10-12 B122.1 Neuvonta
16.05.2023 10-12 B103 Luento
22.05.2023 10-12 C231.1 Neuvonta
25.05.2023 14-18 B103 Tentti
```

Tästä luettelosta on hankala hahmottaa viikkorytmiä. Siitä on myös hankala huomata, että torstaina 27.4.2023 luento oli eri paikassa kuin tavallisesti (koska sali oli varattu johonkin muuhun tilaisuuteen), ja joinakin päivinä opetusta ei ollutkaan (kuten vappu ja helatorstai). Tiedot on helpompi tulkita, jos ne esitetään lukujärjestyksenä ja poikkeusten luettelona kuten kuvassa 1.

Lukujärjestystehtävän vaiheen 2 tavoitteena on ohjelma, joka tuottaa esimerkin mukaisesta tapahtumien luettelosta esimerkin mukaiset lukujärjestyksen ja poikkeusten luettelon. Jotta kaikkea ei tarvitsisi hallita kerralla ja jotta olisi mahdollisuus saada piste vaikka ei tekisi tehtävää kokonaan, tehtävässä on myös vaihe 1.

Tehtävässä ei tarvita erityisiä algoritmeja ja tietorakenteita, vaan perustason ohjelmointikeinot riittävät. Tehtävä voi silti tuntua työläältä, sillä pikkuyksityiskohtien viilaamista on paljon, eikä ole ihan helppoa selvittää, mikä samalle viikonpäivälle ja kellonajalle osuvista tapahtumista merkitään lukujärjestykseen ja mitkä poikkeusten luetteloon, vai merkitäänkö kaikki poikkeusten luetteloon. Pikkuyksityiskohtien saamista kohdalleen on kuitenkin tärkeää harjoitella, sillä se

```
TIEA211 Algoritmit 2 Kevät 2023
21.3.2023-25.5.2023
Poikkeukset:
3.4.2023 10-12 ei tapahtumaa
4.4.2023 10-12 ei tapahtumaa
6.4.2023 14-18 ei tapahtumaa
10.4.2023 10-12 ei tapahtumaa
27.4.2023 14-16 A102 Luento
1.5.2023 10-12 ei tapahtumaa
18.5.2023 14-18 ei tapahtumaa
22.5.2023 10-12 C231.1 Neuvonta
23.5.2023 10-12 ei tapahtumaa
25.5.2023 14-18 B103 Tentti
      | Maanantai   | Tiistai     | Keskiviikko | Torstai     | Perjantai   |
------+-------------+-------------+-------------+-------------+-------------+
 8- 9 |             |             |             |             |             |
------+-------------+-------------+-------------+-------------+-------------+
 9-10 |             |             |             |             |             |
------+-------------+-------------+-------------+-------------+-------------+
10-11 | B122.1 Neuv | B103 Luento |             |             |             |
------+             +             +-------------+-------------+-------------+
11-12 |             |             |             |             |             |
------+-------------+-------------+-------------+-------------+-------------+
12-13 |             |             |             |             |             |
------+-------------+-------------+-------------+-------------+-------------+
13-14 |             |             |             |             |             |
------+-------------+-------------+-------------+-------------+-------------+
14-15 |             |             |             | B103 Luento |             |
------+-------------+-------------+-------------+             +-------------+
15-16 |             |             |             |             |             |
------+-------------+-------------+-------------+-------------+-------------+
16-17 |             |             |             | C231.1 Neuv |             |
------+-------------+-------------+-------------+             +-------------+
17-18 |             |             |             |             |             |
------+-------------+-------------+-------------+-------------+-------------+
```

Kuva 1: Lukujärjestysohjelman tulostus esimerkkisyötteelle

on usein olennaista algoritmien ohjelmoinnissa (ja muutenkin ohjelmistotyössä). Tässä tehtävässä sen harjoittelemista helpottaa se, että monet virheet näkyvät selvästi esimerkiksi liian kapeana tai leveänä tulostuksena.

Ohjelmassa kannattaa käyttää yksi- tai kaksiulotteista taulukkoa, jossa on yhteensä 120 lokeroa, yksi jokaista arkipäivää maanantai–perjantai ja jokaista vuorokauden tuntia varten.

Vaiheessa 1 tuotetaan vain lukujärjestys, käyttäen kunkin tunnin ja viikonpäivän kohdalla syötteessä ensin esiintynyttä tapahtumaa. Pois jää kunkin tunnin ja viikonpäivän eniten esiintyvän vaihtoehdon (jokin tapahtuma tai ei tapahtumaa) tunnistaminen ja muut vaihtoehdot esittävän poikkeusten luettelon tuottaminen. Vaihe 1 ei vastaa todellista lukujärjestyksen tuottamista. Sen tarkoitus on vain olla välivaihe matkalla käyttökelpoiseen ohjelmaan tai kohta, jossa tehtävän tekemisen voi keskeyttää ja silti saada pisteen.

Yksityiskohtien paljouden vuoksi opettajan ohjelma tälle tehtävälle on selvästi pitempi kuin muille tehtäville, vaiheeseen 1 noin 210 ja vaiheeseen 2 noin 340 riviä. Yleisestä linjasta poiketen lukujärjestysohjelman syötteessä voi olla enintään yksi lukujärjestystehtävä.

Tämän ohjelmointitehtävän taustalla on samankaltainen ohjelma, joka tulostaa lukujärjestyksen www-sivuna ja jolle syöte maalataan ja kopidaan sen ohjelman www-sivulta, jolla opetustiloja varataan todellisuudessa. Alla oleva syötteen kuvaus matkii joitakin niin saadun syötteen piirteitä.

Tässä ohjelmointitehtävässä tyhjällä rivillä tarkoitetaan riviä, jossa ei ole mitään tai mitään muuta kuin välilyöntejä. Syötteen ensimmäinen epätyhjä rivi on otsikkorivi. Kukin muu epätyhjä rivi kertoo yhden tapahtuman muodossa v^* *päivämäärä* v^+ *kellonaika* v^* - v^* *kellonaika* v^+ *nimi* v^*, missä v^* tarkoittaa nollaa tai useampaa välilyöntiä ja v^+ tarkoittaa yhtä tai useampaa välilyöntiä. Päivämäärä on muotoa *pp.kk.vvvv* tai *p.kk.vvvv* tai *pp.k.vvvv* tai *p.k.vvvv*. Vuosi on vähintään 2000 ja enintään 2099. Kuukausi on vähintään 1 ja enintään 12, ja päivä on vähintään 1 ja enintään kuukauden viimeinen päivä. Kellonaika on vähintään 0 ja enintään 24. Jälkimmäinen kellonaika on suurempi kuin ensimmäinen. Nimen sisällä voi olla välilyöntejä, mutta nimen edessä ja perässä mahdollisesti olevat välilyönnit eivät kuulu nimeen.

Tapahtumat eivät välttämättä tule aikajärjestyksessä. Mikään tapahtuma ei osu lauantaille eikä sunnuntaille. Samalle ajanhetkelle ei osu useampaa kuin yksi tapahtuma. Kaiken kaikkiaan tapahtumia on enintään kaksisataa.

Tyhjiä rivejä voi olla syötteessä missä tahansa. Merkit on UTF-8-koodattu.

Ohjelma saa luottaa siihen, että syöte noudattaa näitä sääntöjä. Ohjelma saa mutta sen ei tarvitse tuottaa virheellisestä syötteestä virheilmoituksen.

Tulosteessa pitää olla ensin otsikkorivi muuten sellaisenaan, mutta sen alussa ja lopussa mahdollisesti olevat välilyönnit on poistettu. Jos otsikkoriviä ei saatu, niin pitää tulostaa rivi jossa lukee ”Syöte puuttuu” ja lopettaa. Sitten pitää olla rivi,

jossa on kurssin ajanjakso muodossa päivämäärä-päivämäärä tai ”Ei tapahtumia”. Päivämäärät ovat varhaisimman ja myöhäisimmän tapahtuman päivämäärät. Vaiheessa 2 seuraavaksi tulostetaan mahdolliset poikkeukset lukujärjestyksestä. Jos niitä on, niin ensin on omalla rivillään ”Poikkeukset:” ja sitten poikkeukset aikajärjestyksessä kukin omalla rivillään muodossa päivämäärä välilyönti kellonaika-kellonaika välilyönti nimi tai ”ei tapahtumaa”. Aina kun mahdollista, poikkeus pitää tulostuksessa yhdistää edelliseen poikkeukseen muuttamalla sen loppumisaikaa.

Sekä vaiheessa 1 että vaiheessa 2 tulostetaan lopuksi lukujärjestys. Sen merkit ja mitat selviävät kuvan 1 esimerkistä.

Lukujärjestykseen tulostetaan rivi jokaiselle tunnille alkaen päivän ensimmäisestä tapahtumasta tai tunnista 8–9 sen mukaan kumpi on varhaisempi, ja päättyen päivän viimeiseen tapahtumaan tai tuntiin 15–16 sen mukaan kumpi on myöhäisempi.

Lukujärjestykseen tulostetaan kunkin tunnin ja viikonpäivän (paitsi lauantain ja sunnuntain) kohdalle niitä vastaava tieto kuitenkin siten, että jos epätyhjä tieto on sama kuin lähinnä ylemmässä lokerossa, niin lokeroita erottava viivanpätkä jätetään tulostamatta ja lokero jätetään tyhjäksi. Vaiheessa 1 tuntia ja viikonpäivää vastaava tieto on syötteessä ensimmäisenä sille tunnille ja viikonpäivälle osuvan tapahtuman nimi. Vaiheessa 2 se on tunnin ja viikonpäivän kohdalla eniten esiintyvä nimi tai tyhjä, jos tyhjä esiintyy useammin kuin mikään nimi. Jos eniten esiintyvä tieto ei ole yksikäsitteinen, niin tulostetaan se eniten esiintyvä epätyhjä tieto, jolla on varhaisin esiintymispäivä.

Jos tieto on pitempi kuin 11 UTF-8-merkkiä, niin tulostetaan sen ensimmäiset 11 UTF-8-merkkiä. Tulostus täydennetään välilyönneillä riittävän pitkäksi. Jos lukujärjestykseen tulisi vain tyhjiä ruutuja, sen sijaan tulostetaan rivi jossa lukee ”Ei muita tapahtumia”.

Päivämäärät ja kellonajat tulostetaan ilman etunollia. Edellä mainittua vapaaehtoista virheilmoitusta lukuun ottamatta tulosteessa ei saa olla yhtään ylimääräistä merkkiä, ei edes välilyöntejä eikä rivinsiirtoja.

4 Tehtävä: Aikataulu

Ohjelman tehtävänä on laatia aikataulu esimerkiksi parturille tai esitelmille, jotka opiskelijat pitävät jollakin kurssilla.

Asiakkaat ilmoitetaan peräkkäisinä lukuina ykkösestä alkaen ja ajat lukuina väliltä 1, ..., 999. Syötteessä on nolla tai useampia testitapauksia ja lopuksi 0. Testitapauksessa on asiakkaalle 1 sopivat ajat, asiakkaalle 2 sopivat ajat ja niin edelleen, ja lopuksi 0. Asiakkaalle sopivien aikojen perässä on 0. (Asiakkaalle on pakko ilmoittaa ainakin yksi sopiva aika, koska muuten asiakas ei erotu testitapauksen tai syötteen lopusta.) Jokaisen kahden tekstialkion välissä on ainakin yksi välilyönti ja/tai rivinsiirto, minkä lisäksi alussa ja lopussa saa olla ylimääräisiä välilyöntejä ja/tai rivinsiirtoja. Ohjelma saa luottaa siihen, että syöte on kuvauksen mukainen. Kuitenkin, jos haluat, niin saat laittaa ohjelmasi ilmoittamaan lupausten vastaisesta syötteestä. Esimerkki:

```
3 1 2 0
3 0
3 1 0
0
4 2 5 0 2 4 0 4 2 0  0
1 2 4 0   1 0   1 3 5 0   4 0   1 4 0  0
                                       0
```

Ohjelman tulee tulostaa aikataulu, jossa mahdollisimman monelle asiakkaalle on annettu hänelle sopiva aika, eikä mitään aikaa ole annettu useammalle kuin yhdelle asiakkaalle. Ensin tulostetaan asiakkaan 1 saama aika tai 0 merkiksi, että hän ei saanut aikaa; sitten asiakkaan 2 saama aika tai 0 ja niin edelleen. Mikään tulostusrivi ei saa olla yli 79 merkkiä pitkä, ja jokaisen kahden tekstialkion välissä on oltava ainakin yksi välilyönti ja/tai rivinsiirto. Muuten välilyöntien ja rivinsiirtojen käyttö on vapaata.

Jos on mahdotonta antaa jokaiselle aika, niin pitää suosia pienempinumeroisia asiakkaita. Täsmällisemmin sanoen, lopputuloksessa ei saa olla sellaisia asiakkaita A ja B, että A:n numero on pienempi kuin B:n, A jäi ilman aikaa, ja B sai ajan joka olisi sopinut myös A:lle.

Tulostuksen tulkitsemisen helpottamiseksi kukin aika tai 0 kannattaa (mutta ei ole pakko) tulostaa 4 merkkiä leveän kentän oikeaan reunaan. Kullekin riville kannattaa tulostaa 10 aikaa tai nollaa, paitsi viimeinen rivi saa jäädä vajaaksi. Myös viimeisen rivin loppuun kannattaa tulostaa rivinsiirto, vaikka se olisi vajaa.

Edellisestä esimerkistä voi tulla tällainen tulos:

```
   2   3   1
   5   2   4
   2   1   3   4   0
```

Keskimmäisenä rivinä voi olla myös `5 4 2`, sillä siinäkin jokainen saa hänelle sopivan ajan eikä mitään aikaa ole annettu kahdelle. Viimeinen rivi voi olla myös `2 1 5 4 0`. Sen sijaan viimeinen rivi ei voi olla `2 0 3 4 1`, koska siinä asiakas 2 jäi ilman aikaa mutta hänelle sopiva aika 1 oli annettu suurempinumeroiselle asiakkaalle 5.

Ohjelma kannattaa toteuttaa siten, että se pitää koko ajan yllä aikataulua, jossa mahdollisimman moni siihen mennessä käsitelty asiakas on saanut ajan. Seuraavan asiakkaan käsittely kannattaa ajatella suunnattua graafia koskevana tehtävänä. Olkoon tämä asiakas F. Jos on olemassa sellaiset ajan jo saaneet asiakkaat A, B, C, D ja E, että A:lle voidaan antaa vapaa aika, A:lta vapautunut aika voidaan antaa B:lle, B:ltä vapautunut aika voidaan antaa C:lle ja niin edelleen, ja E:ltä vapautunut aika voidaan antaa F:lle, niin F saadaan lisättyä aikatauluun. Voidaan todistaa (mutta ei todisteta tässä), että muussa tapauksessa ei ole mahdollista antaa F:lle aikaa ilman että joku ajan jo saanut menettää aikansa eikä saa uutta tilalle.

Opettajan malliohjelma on juuri ja juuri yli 100 riviä pitkä.

5 Tehtävä: Rivijako

Tehtävänä on laatia ohjelma, joka jakaa tekstin riveille niin, että kunkin tekstikappaleen lyhyin muu kuin viimeinen rivi on mahdollisimman pitkä. Vaiheessa 1 toteutetaan kaikki muu toiminnallisuus paitsi riveille jakamisen optimointi. Vaiheessa 2 ohjelma muutetaan jakamaan riveille optimaalisesti.

Syötteenä on jono UTF-8-merkkejä. Syöte muodostuu sanoista. Sanassa voi olla mitä tahansa muita merkkejä kuin välilyöntejä ja rivinsiirtoja. Alussa, sanojen välissä sekä lopussa voi olla miten paljon tahansa välilyöntejä ja rivinsiirtoja kuitenkin siten, että kahden peräkkäisen sanan välissä on ainakin yksi välilyönti tai rivinsiirto. Kaksi peräkkäistä rivinsiirtoa lopettaa testitapauksen. Myös syötteen loppuminen lopettaa testitapauksen.

Kukin testitapaus, jossa on ainakin yksi sana, pitää pilkkoa riveihin seuraavasti:

- Sanat tulostetaan siinä järjestyksessä kuin ne tulevat syötteessä.
- Kahden peräkkäisen sanan välissä on joko yksi välilyönti tai yksi rivinsiirto. Lisäksi testitapauksen lopussa on rivinsiirto. Muita välilyöntejä ja rivinsiirtoja ei ole.
- Mikään rivi, jolla on useampi kuin yksi sana, ei ole pitempi kuin 30 merkkiä.
- Vaiheessa 1 jokaiselle riville tulostetaan niin monta sanaa kuin mahtuu. Toisin sanoen, epätyhjä rivi katkaistaan vasta kun välilyönti ja seuraava sana eivät mahtuisi sille. Yli 30 merkkiä pitkä sana ei mahdu epätyhjän rivin jatkoksi, mutta mahtuu yksinään omalle rivilleen.
- Vaiheessa 2
  - Lyhyin muu kuin viimeinen rivi on mahdollisimman pitkä.
  - Siinä määrin kuin mahdollista huonontamatta edellistä kriteeriä, rivejä on mahdollisimman vähän.
  - Siinä määrin kuin mahdollista huonontamatta edellisiä kriteerejä, viimeinen rivi on mahdollisimman lyhyt.

Testitapauksesta, jossa ei ole yhtään sanaa, ei tulosteta mitään. Kahden epätyhjän testitapauksen väliin tulostetaan täysin tyhjä rivi. (Tässä täysin tyhjä rivi tarkoittaa riviä, jolla ei ole mitään, ei edes välilyöntejä.)

Tässä on esimerkki syötteestä. Jos kopioit sen siitä koeajaaksesi ohjelmaasi, niin ota huomioon, että rivinsiirrot ja ylimääräiset välilyönnit saattavat kadota kopioitaessa. Ennen koeajoa lisää ainakin kadonneet rivinsiirrot sekä välilyönti jälkimmäiselle tyhjältä näyttävälle riville.

```
Testitapaus, jossa ahne algoritmi ei_tuota_parasta_tulosta.

Seuraavalla rivillä on välilyönti.

Tiedäthän, että vaikka ä on kaksi tavua, se on vain yksi merkki.
  Merkki voi      muodostua jopa 4 tavusta.
 Jätä laskuista pois tavut muotoa 10xxxxxx.
```

Vaiheessa 1 siitä pitää tulostua seuraava:

```
Testitapaus, jossa ahne
algoritmi
ei_tuota_parasta_tulosta.

Seuraavalla rivillä on
välilyönti. Tiedäthän, että
vaikka ä on kaksi tavua, se on
vain yksi merkki. Merkki voi
muodostua jopa 4 tavusta. Jätä
laskuista pois tavut muotoa
10xxxxxx.
```

Vaiheessa 2 tulostuksen pitää alkaa seuraavavasti. Loput tulostuksesta on samoin kuin vaiheessa 1.

```
Testitapaus, jossa
ahne algoritmi
ei_tuota_parasta_tulosta.
```

Vaihe 1 on perusohjelmointia. Opettajan malliohjelma siihen on noin 80 riviä pitkä.

Vaiheessa 2 ohjelma kannattaa laittaa kunkin sanan kohdalla etsimään ja tallettamaan paras lyhyimmän rivin pituus, jos sanoja otettaisiin huomioon vain vuorossa olevaa sanaa edeltävään sanaan saakka ja myös viimeinen rivi otettaisiin huomioon mahdollisena lyhyimpänä rivinä. Se onnistuu kokeilemalla tapaukset, joissa vuorossa olevaa edeltävä sana on viimeisellä rivillä yksinään, yhdessä sitä edeltävän sanan kanssa, kahden edeltävän sanan kanssa ja niin edelleen. Viimeisen rivin pituus on helppo laskea. Paras muiden kuin viimeisen rivin pituus on talletettu viimeisen rivin ensimmäisen sanan yhteyteen. Tapauksen tulos on niistä pienempi. Kokeiltavien tapausten tuloksista paras talletetaan vuorossa olevan sanan yhteyteen.

Koko tekstikappaleen viimeisen rivin alkukohta saadaan samankaltaisella eri tapausten kokeilemisella. Molempia ohjelman osia täytyy hieman täydentää ottamaan huomioon, että jos paras lyhyimmän rivin pituus voidaan saavuttaa usealla

tavalla, niin niiden joukosta pitää valita sellainen, joka minimoi rivien määrän. Sen jälkeen onkin jo aika miettiä, miten ohjelma saataisiin tulostamaan sanat niin että rivit on katkaistu niistä kohdista, jotka tuottavat parhaan tuloksen.

Opettajan malliohjelma on noin 110 riviä pitkä.

6 Tehtävä: Esitiedot

Ohjelman tehtävänä on tulostaa kursseille suoritusjärjestys tai kurssien esitietoriippuvuuksissa oleva silmukka. Vuodessa on neljä periodia. Kurssin suorittaminen kestää yhden periodin. Kukin kurssi opetetaan täsmälleen kerran vuodessa, ja joka vuosi samalla periodilla. Kurssin saa suorittaa vasta kun sen jokainen esitietokurssi on suoritettu. Yhtäaikaa saa suorittaa miten monta kurssia tahansa. Jokainen kurssi pitää suorittaa mahdollisimman aikaisin.

Syötteenä on nolla tai useampia testitapauksia ja lopuksi 0. Testitapaus sisältää yhden tai useamman kurssin ja lopuksi 0. Kurssin tiedot ovat muotoa *numero nimi periodi* : *esitiedot* 0. Numero on juokseva numero ykkösestä alkaen. Nimi on merkkijono, joka ei sisällä välilyöntejä eikä rivinsiirtoja. Periodi on 1, 2, 3 tai 4. Esitiedot on nolla tai useampi kurssin numero. Alussa, lopussa ja minkä tahansa kahden tekstialkion välissä voi olla vapaasti välilyöntejä ja/tai rivinsiirtoja. Minkä tahansa kahden tekstialkion välissä on ainakin yksi välilyönti tai rivinsiirto.

Ohjelma saa luottaa siihen, että syöte on kuvauksen mukainen. Kuitenkin, jos haluat, niin saat laittaa ohjelmasi ilmoittamaan lupausten vastaisesta syötteestä.

Esimerkki:

```
1 Sähly 1: 7 3 8 0
2 Pitsinnypläys 4: 5 4 8 6 1 10
  7 3 0
3 Ruoanlaitto 1: 7 8 0
4 Pakohuone 1: 7 5 1 8 3 0
5 Roolipeli 2: 8 7 3 1 0
6 Suunnistus 3: 4 10 8 5 1 3 7 0
7 Hikijumppa 2: 8 0
8 Elokuvakerho 3: 0
9 Auringonotto 1: 8 3 6 10 5 2 4
  1 7 0
10 Rumpujen_soitto 3: 3 1 4 7 8
  5 0 0

1 Lähdekritiikki 1: 0 2 Etiikka 2: 6 5 0
3 Tutkimusmenetelmät 3: 2 8 4 5 0 4 Tietosuoja 4: 0
5 Filosofia 1: 7 6 0 6 Äidinkieli 2

:

0 7 Diskurssianalyysi 3: 2 6 1 0 8 Tilastotiede 4: 0
0

0
```

Jos testitapauksessa on silmukka, niin ohjelman pitää tulostaa rivi, jossa lukee ”Silmukka:”. Sen jälkeen ohjelman pitää tulostaa rivejä muotoa *numero nimi*, missä *numero* on jonkin kurssin numero ja *nimi* on saman kurssin nimi, ensimmäisenä tulostettu kurssi on viimeisenä tulostetun kurssin esitieto, kukin muu tulostettu kurssi on edellisenä tulostetun kurssin esitieto, eikä mitään kurssia tulosteta useammin kuin kerran. Numeron ja nimen edessä saa olla vapaasti välilyöntejä. Nimen edessä pitää olla ainakin yksi välilyönti, jollei se muuten erotu numerosta. Lopuksi ohjelman pitää tulostaa täysin tyhjä rivi. Mitään muuta ohjelma ei saa tulostaa.

Jos testitapauksessa ei ole silmukkaa, ohjelman pitää tulostaa rivi, jossa lukee ”Suoritusajankohdat:”. Sen jälkeen ohjelman pitää tulostaa kutakin kurssia kohden täsmälleen yksi rivi muotoa *ajankohta* : *numero nimi*, missä *numero* on kurssin numero ja *nimi* on saman kurssin nimi. Kurssin ajankohdan pitää olla pienin positiivinen kokonaisluku, joka on kurssin periodi plus jokin luvun 4 monikerta, ja jolle kurssin jokainen esitietokurssi on suoritettu jonain aikaisempana ajankohtana. Rivien pitää olla ajankohtien mukaisessa kasvavassa järjestyksessä. Tekstialkioiden edessä saa olla vapaasti välilyöntejä. Nimen edessä pitää olla ainakin yksi välilyönti, jollei se muuten erotu numerosta. Lopuksi ohjelman pitää tulostaa täysin tyhjä rivi. Mitään muuta ohjelma ei saa tulostaa.

Edellä olleesta syötteestä tulee esimerkiksi seuraavanlainen tulostus:

```
Suoritusajankohdat:
3: 8 Elokuvakerho
6: 7 Hikijumppa
9: 3 Ruoanlaitto
13: 1 Sähly
14: 5 Roolipeli
17: 4 Pakohuone
19: 10 Rumpujen_soitto
23: 6 Suunnistus
24: 2 Pitsinnypläys
25: 9 Auringonotto

Silmukka:
2 Etiikka
5 Filosofia
7 Diskurssianalyysi
```

Tehtävän voi ratkaista tehokkaasti syvyyshaulla.

Vaihtoehtoisesti tehtävän voi ratkaista tehokkaasti ylläpitämällä kullekin periodille niiden kurssien joukkoa, joiden kaikki esitiedot on suoritettu. Jokaisessa

kurssissa on suorittamattomien esitietokurssien määrä sekä linkit niihin kursseihin, joiden esitieto se on. Kun kurssi suoritetaan, löydetään näiden linkkien avulla nopeasti ne kurssit, joiden suorittamattomien esitietokurssien määrää pitää vähentää.

Opettajan malliohjelma on noin 150 riviä pitkä.

7 Esitehtävä: Presidentinvaali

Tehtävänä on laatia ohjelma, joka selvittää presidentinvaalin ensimmäisen kierroksen äänimääristä voittajan tai toiselle kierrokselle pääsijät.

Syötteenä on ehdokkaiden tiedot. Kunkin ehdokkaan tiedot ovat omalla rivillään seuraavasti. Rivin alussa voi olla nolla tai useampi välilyönti. Sitten on ehdokkaan saama äänimäärä etumerkittömänä kokonaislukuna, vähintään yksi välilyönti, ehdokkaan nimi sekä nolla tai useampi välilyönti. Nimi voi olla esimerkiksi ”Kelju K. Kojootti”. Nimi koostuu yhdestä tai useammasta epätyhjästä osasta. Nimen osissa ei ole välilyöntejä. Osien välissä on ainakin yksi välilyönti. Ohjelma saa luottaa siihen, että syöte noudattaa tätä kuvausta, ja että ehdokkaita on ainakin kaksi.

Tässä on esimerkki syötteestä:

```
 572733   Abraham Ahkera
      0   Ei Saanut Ääntä   Edes  Itseltään!
2912229   Kaisa  Kansansuosikki
     41   Kelju K. Kojootti
                                      48622 Oskari Oikea-Reunanen
 572733   Pirkko Poliitikko
572633 Reetta Vasen-Reunanen
 572733   T. Tasaääninen
572733      Toisinajattelija    Välilyöntinen
```

Jos joku ehdokas saa enemmän kuin puolet kaikista äänistä, tulee ohjelman tulostaa ”Onnea presidentti *nimi*!”. Moniosaisen nimen osien väliin pitää tulostaa täsmälleen yksi välilyönti. Yllä olevalla syötteellä ohjelman pitää tulostaa

```
Onnea presidentti Kaisa Kansansuosikki!
```

Muussa tapauksessa ohjelman tulee tulostaa ensin rivi ”Toiselle kierrokselle:” ja sen jälkeen jokaisesta toiselle kierrokselle päässeestä rivi, jossa on hänen nimensä eikä mitään muuta. Toiselle kierrokselle päässeet tulee tulostaa siinä järjestyksessä, kuin he esiintyivät syötteessä. Ehdokas pääsee toiselle kierrokselle jos ja vain jos korkeintaan yksi ehdokas sai enemmän ääniä kuin hän. Jos edellä olleesta esimerkkisyötteestä vähennetään Kaisa Kansansuosikin äänimäärää yhdellä, niin ohjelman pitää tulostaa

```
Toiselle kierrokselle:
Abraham Ahkera
Kaisa Kansansuosikki
Pirkko Poliitikko
T. Tasaääninen
Toisinajattelija Välilyöntinen
```

Opettajan tekemä ohjelma on noin 60 riviä pitkä.

8 Esitehtävä: Sähköpostien järjestäminen

Ohjelman tehtävänä on järjestää sähköpostiviestien tiedot ensisijaisesti aiheen ja toissijaisesti kellonajan mukaiseen järjestykseen, ja tulostaa ne helppolukuisessa muodossa. Jos kahdella viestillä on sama aihe ja sama kellonaika, niin niiden alkuperäinen järjestys on säilytettävä. Aiheen alussa mahdollisesti olevia ”Re: ”, ”[SPAM] ” ja muita sellaisia ei oteta huomioon aiheen mukaan järjestettäessä.

Järjestämisalgoritmi on *vakaa* (*stable*), jos ja vain jos se ei koskaan vaihda kahden sellaisen alkion järjestystä, joilla on sama avain. Ohjelmointikielissä on valmiina vakaita järjestämisalgoritmeja. Myös epävakaata järjestämisalgoritmia voidaan käyttää antamalla syöteriveille juokseva numero ja ottamalla se huomioon ylimääräisenä, vähiten merkitsevänä järjestyskriteerinä.

Tässä tehtävässä *sana* tarkoittaa mitä tahansa mahdollisimman pitkää yhtenäistä syötteen osaa. Toisin sanoen, sana on aivan syöterivin alussa tai sen edessä on välilyönti, sanan sisällä ei ole välilyöntejä eikä rivinsiirtoja, ja sana on aivan syöterivin lopussa tai sen perässä on välilyönti.

Syötteen ensimmäinen rivi luettelee ne sanat, joita ei oteta huomioon aiheen alussa aiheen mukaan järjestettäessä. Esimerkiksi jos ensimmäinen rivi on ”Re: Vs:”, niin aiheet ”Re: Heippa!”, ”Vs: Heippa!”, ”Vs: Vs: Re: Heippa!” ja ”Heippa!” katsotaan vertailussa samaksi aiheeksi. Sen sijaan ”Re:Heippa!” on eri aihe, koska siinä ”Re:” ei ole sana. Sekä ”Vs: ”, ”Vs:” että ”” tarkoittavat tyhjää aihetta. Ohjelma saa luottaa siihen, että syötteen ensimmäisellä rivillä on vähän sanoja, ja ne ovat lyhyitä.

Jokainen muu syöterivi on joko täysin tyhjä (eli siinä ei ole mitään, ei edes välilyöntejä) tai siinä on ensin viestin lähettäjä, sitten ”|”, sitten viestin aihe, sitten ”|”, ja lopuksi kellonaika.

Lähettäjän ja aiheen alussa ja lopussa olevat välilyönnit poistetaan. Sanojen väliin jätetään yksi välilyönti ja muut poistetaan.

Kellonaika on joko muotoa *hh*:*mm*:*ss* tai muotoa *h*:*mm*:*ss*. Aiheen lopettavan ”|” ja kellonajan välissä on nolla tai useampia välilyöntejä. Myös rivin lopussa on nolla tai useampia välilyöntejä.

Syötteessä ei ole muita merkkejä kuin näkyviä Ascii-merkkejä ja rivinsiirtoja. Siis esimerkiksi ääkkösiä ei ole. Suuruusjärjestyksen vertailussa tulee käyttää Ascii-merkkien mukaista järjestystä. Monissa ohjelmointikielissä käytetään juuri sitä, ellei ohjelmassa erikseen käsketä toisin.

Ohjelma saa luottaa siihen, että syöte noudattaa näitä sääntöjä. Jos syöte rikkoo niitä, niin ohjelma saa, mutta sen ei ole pakko, tulostaa vapaamuotoisen virheilmoituksen.

Tässä on esimerkki syötteestä:

```
           [SPAM]          Re:     Vs:  /  //
Assari, Ahkera-Liisa|Vs: Milloin on seuraava tentti?| 10:01:28
    Oiva Opiskelija   |    Milloin on seuraava tentti?|9:55:05
Opettaja, Kuka Lie | Viikon 7 tehtavat | 09:20:23

Tarmokas Tietava   | Aihe alkaa aa:lla | 14:03:55


Einari Ei-Viivaa|        Mimmoinen viesti|8:51:06
Jaakko Jakoviiva| /      Mimmoinen viesti|11:10:38
Kaarina Kaksoisviiva|//  Mimmoinen viesti|11:03:02
Reku Re Viiva|  Re:  /   Mimmoinen viesti| 9:33:51
```

Kunkin sähköpostiviestin tiedot tulostetaan omalle rivilleen. Rivillä on ensin lähettäjä, sitten aihe ja lopuksi kellonaika.

Lähettäjä tulostetaan kenttään, jonka leveys on sama kuin pisimmän syötteessä esiintyneen lähettäjän mutta enintään 25 merkkiä. Jos lähettäjä on pitempi kuin 25 merkkiä, niin tulostetaan vain ensimmäiset 25 merkkiä. Lähettäjän perään tulostetaan välilyöntejä kunnes kentän leveys on saavutettu. Tämän jälkeen tulostetaan vielä yksi välilyönti. (Se varmistaa, että lähettäjän ja aiheen välissä on ainakin yksi välilyönti.)

Aihe tulostetaan kenttään, jonka leveys on sama kuin pisimmän syötteessä esiintyneen aiheen mutta enintään sellainen, että lähettäjän kenttä, sen perässä oleva välilyönti ja aiheen kenttä ovat yhteensä enintään 70 merkkiä leveät. Jos aihe ei mahdu kenttäänsä, niin tulostetaan vain sen alkua niin paljon kuin mahtuu (mutta järjestämisessä otetaan huomioon kaikki merkit). Aiheen perään tulostetaan välilyöntejä kunnes kentän leveys on saavutettu. Tämän jälkeen tulostetaan vielä yksi välilyönti.

Jos kellonajan tunti on vähintään 10, niin kellonaika tulostetaan muodossa *hh*:*mm*:*ss*. Jos tunti on alle 10, niin kellonaika tulostetaan muuten samassa muodossa, mutta tunnin ensimmäisen numeron paikalle tulostetaan välilyönti.

Mitään muuta ohjelma ei saa tulostaa.

Edellä olleesta syötteestä pitää tulla tällainen tuloste:

```
Tarmokas Tietava     Aihe alkaa aa:lla               14:03:55
Oiva Opiskelija      Milloin on seuraava tentti?      9:55:05
Assari, Ahkera-Liisa Vs: Milloin on seuraava tentti? 10:01:28
Einari Ei-Viivaa     Mimmoinen viesti                 8:51:06
Reku Re Viiva        Re: / Mimmoinen viesti           9:33:51
Kaarina Kaksoisviiva // Mimmoinen viesti             11:03:02
Jaakko Jakoviiva     / Mimmoinen viesti              11:10:38
Opettaja, Kuka Lie   Viikon 7 tehtavat                9:20:23
```

Opettajan tekemä ohjelma tälle tehtävälle on noin 120 riviä pitkä.

Liite A Syötteen lukeminen

Syötteen lukeminen vaihtelee eri ohjelmointikielissä. Tässä liitteessä esitellään kolmen eri ohjelmointikielen syötteenlukutoimintoja kurssin ohjelmointitehtävien tekemiseen riittävässä määrin.

Java. Syöte kannattaa lukea kirjaston `Scanner` avulla. Se saadaan käyttöön kirjoittamalla lähelle ohjelman alkua

```
import java.util.Scanner;
```

Syötteen lukemista varten pitää luoda yksi `Scanner` -olio. Sille voi antaa haluamansa nimen. Näin voidaan luoda `Scanner` -olio nimeltä `syote` :

```
Scanner syote = new Scanner( System.in );
```

Syötteestä voi lukea kokonaisluvun toiminnolla `syote.nextInt()` ja merkkijonon toiminnolla `syote.next()` . Niitä voi käyttää esimerkiksi seuraavasti:

```
int luku = syote.nextInt();
String merkkijono = syote.next();
```

Kuvassa 2 on ohjelma, joka lukee syötteestä vuorotellen luvun ja merkkijonon ja tulostaa ne, kunnes lukuna on nolla. Syötteen loppumerkkinä toimivaa nollaa ei tulosteta. Riippumatta siitä, miten syöte on jaettu riveihin, kukin luku ja siihen liittyvä merkkijono tulostetaan samalle riville. Niiden väliin tulostetaan täsmälleen yksi välilyönti. Mitään muuta ohjelma ei tulosta.

Osassa kurssin ohjelmointitehtäviä ei tarvita muita syötteenkäsittelytoimintoja kuin edellä mainitut. Niissä syötteen lukeminen voidaan toteuttaa kuvan 2 havainnollistamilla keinoilla.

```
import java.util.Scanner;
public class lukuja_ja_merkkijonoja{
  public static void main( String[] args ){
    Scanner syote = new Scanner( System.in );
    while( true ){
      int luku = syote.nextInt();
      if( luku == 0 ){ break; }
      String merkkijono = syote.next();
      System.out.println( luku + " " + merkkijono );
    }
  }
}
```

Kuva 2: Java esimerkki lukujen ja merkkijonojen lukemisesta

```
import java.util.Scanner;
public class kopioija{
  public static void main( String[] args ){
    Scanner syote = new Scanner( System.in );
    while( syote.hasNextLine() ){
      String rivi = syote.nextLine();
      boolean tyhja = true;
      for( int i = 0; tyhja && i < rivi.length(); ++i ){
        tyhja &= rivi.charAt(i) == ' ';
      }
      if( tyhja ){ System.out.println( "Tyhjä rivi" ); }
      else{ System.out.println( rivi ); }
    }
  }
}
```

Kuva 3: Java esimerkki riveittäin lukemisesta

Kuvan 2 ohjelma luottaa siihen, että syötteessä tulee luku silloin kun se haluaa lukea luvun ja syötteessä tulee merkkijono silloin kun se haluaa lukea merkkijonon. Ohjelmia käytettäessä ja ohjelmointivirheitä etsittäessä on kuitenkin usein hyödyllistä, jos ohjelma kertoo, että se ei saanut sellaista syötettä kuin oletti saavansa. Tämä voidaan toteuttaa lisäämällä ennen kokonaisluvun lukemista

```
if( !syote.hasNextInt() ){
  System.out.println( "Luku puuttuu!" ); break;
}
```

ja ennen merkkijonon lukemista

```
if( !syote.hasNext() ){
  System.out.println( "Merkkijono puuttuu!" ); break;
}
```

Tällaisten lisäysten tekeminen on vapaaehtoista. Niitä ei vaadita kurssin ohjelmointitehtävissä.

Osassa kurssin ohjelmointitehtäviä syöte kannattaa lukea rivi kerrallaan. Rivin voi lukea toiminnolla `nextLine()` . Jos rivistä on jo luettu alku, `nextLine()` palauttaa loput rivistä. Toiminto `nextLine()` lukee myös rivin lopussa olevan rivinsiirron `'\n'` , mutta ei sisällytä sitä palautettuun merkkijonoon.

```
String rivi = syote.nextLine();
```

Toiminnolla `hasNextLine()` voi testata, onko syötteessä enää rivejä jäljellä. Sen ero toimintoon `hasNext()` on siinä, että `hasNextLine()` huomaa ja `hasNext()` ei huomaa, jos syötteen lopussa on tyhjiä rivejä. Tässä yhteydessä tyhjä rivi tarkoittaa riviä, jolla on vain välilyöntejä tai ei edes niitäkään. Kuvassa 3 on ohjel-

```
#include <iostream>
#include <string>
int main(){
  while( true ){
    int luku = 0; std::cin >> luku;
    if( luku == 0 ){ break; }
    std::string merkkijono; std::cin >> merkkijono;
    std::cout << luku << " " << merkkijono << '\n';
  }
}
```

Kuva 4: C++ esimerkki lukujen ja merkkijonojen lukemisesta

ma, joka kopioi kaiken syötteen tulosteeksi siten, että kunkin tyhjän rivin sijaan se tulostaa rivin, jossa lukee `"Tyhjä rivi"`.

Toisinaan suositellaan sulkemaan `Scanner`-olio kun kaikki syöte on luettu. Sen voi tehdä tyyliin `syote.close();`. Se ei kuitenkaan ole välttämätöntä.

C++. Syötteen lukeminen ja tulostaminen tarvitsevat kirjaston `iostream`. Se saadaan käyttöön kirjoittamalla lähelle ohjelman alkua

```
#include <iostream>
```

Syötteestä voi lukea kokonaisluvun toiminnolla `std::cin >> luku;` ja merkkijonon toiminnolla `std::cin >> merkkijono;`.

Kuvassa 4 on ohjelma, joka lukee syötteestä vuorotellen luvun ja merkkijonon ja tulostaa ne, kunnes lukuna on nolla. Syötteen loppumerkkinä toimivaa nollaa ei tulosteta. Riippumatta siitä, miten syöte on jaettu riveihin, kukin luku ja siihen liittyvä merkkijono tulostetaan samalle riville. Niiden väliin tulostetaan täsmälleen yksi välilyönti. Mitään muuta ohjelma ei tulosta.

Osassa kurssin ohjelmointitehtäviä ei tarvita muita syötteenkäsittelytoimintoja kuin edellä mainitut. Niissä syötteen lukeminen voidaan toteuttaa kuvan 4 havainnollistamilla keinoilla.

Kuvan 4 ohjelma luottaa siihen, että syötteessä tulee luku silloin kun se haluaa lukea luvun ja syötteessä tulee merkkijono silloin kun se haluaa lukea merkkijonon. Ohjelmia käytettäessä ja ohjelmointivirheitä etsittäessä on kuitenkin usein hyödyllistä, jos ohjelma kertoo, että se ei saanut sellaista syötettä kuin oletti saavansa. Tämä voidaan toteuttaa lisäämällä kokonaisluvun lukemisen perään

```
if( !std::cin ){ std::cout << "Luku puuttuu!\n"; break; }
```

ja merkkijonon lukemisen perään

```
if( !std::cin ){ std::cout << "Merkkijono puuttuu!\n"; break; }
```

```
#include <iostream>
#include <string>
int main(){
  while( true ){
    std::string rivi; std::getline( std::cin, rivi );
    if( !std::cin ){ break; }
    bool tyhja = true;
    for( unsigned i = 0; tyhja && i < rivi.size(); ++i ){
      tyhja &= rivi[i] == ' ';
    }
    if( tyhja ){ std::cout << "Tyhjä rivi" << '\n'; }
    else{ std::cout << rivi << '\n'; }
  }
}
```

Kuva 5: C++ esimerkki riveittäin lukemisesta

Tällaisten lisäysten tekeminen on vapaaehtoista. Niitä ei vaadita kurssin ohjelmointitehtävissä.

Osassa kurssin ohjelmointitehtäviä syöte kannattaa lukea rivi kerrallaan. Rivin voi lukea toiminnolla `std::getline( std::cin, rivi );` . Jos rivistä on jo luettu alku, `getline` palauttaa loput rivistä. Toiminto `getline` lukee myös rivin lopussa olevan rivinsiirron `'\n'` , mutta ei sisällytä sitä palautettuun merkkijonoon.

Syötteen loppumisen voi tunnistaa yrittämällä lukemistoimintoa ja testaamalla sen jälkeen, onnistuiko lukeminen eli onko `std::cin` tosi. Kuvassa 5 on ohjelma, joka kopioi kaiken syötteen tulosteeksi siten, että kunkin tyhjän rivin sijaan se tulostaa rivin, jossa lukee `"Tyhjä rivi"` . Tässä yhteydessä tyhjä rivi tarkoittaa riviä, jolla on vain välilyöntejä tai ei edes niitäkään.

C#. Monissa ohjelmointikielissä on kokonaisluvun lukutoiminto, joka tarvittaessa ohittaa välilyöntejä ja rivinsiirtoja löytääkseen sen kokonaisluvun, joka on tarkoitus lukea. Niissä on yleensä myös vastaavanlainen merkkijonon lukutoiminto. Ohjelmointikielessä C# ei ole sellaisia valmiina. Koska ne ovat käteviä joissakin ohjelmointiharjoituksissa, on kuvassa 6 näytetty, miten ne voi toteuttaa. Ne saa kopioida kuvasta ja käyttää omassa ohjelmassa.

Osassa kurssin ohjelmointitehtäviä ei tarvita muita syötteenkäsittelytoimintoja kuin kuvassa 6 esitetty luokka. Kuvassa 7 on esimerkki sen käytöstä. Siinä on ohjelma, joka lukee syötteestä vuorotellen luvun ja merkkijonon ja tulostaa ne, kunnes lukuna on nolla. Syötteen loppumerkkinä toimivaa nollaa ei tulosteta. Riippumatta siitä, miten syöte on jaettu riveihin, kukin luku ja siihen liittyvä merkkijono tulostetaan samalle riville. Niiden väliin tulostetaan täsmälleen yksi välilyönti. Mitään muuta ohjelma ei tulosta.

Kuvan 7 ohjelma luottaa siihen, että syötteessä tulee luku silloin kun se haluaa

```
using System;
using System.Text;

/* Yksinkertainen luokka lukemaan standardisyötteestä
  seuraavan kokonaisluvun tai seuraavan merkkijonon. */
class Lukija{
  private char viim_mrk = '\n';

  /* Ohittaa tyhjän tilan standardisyötteestä ja
    palauttaa tiedon onko syötettä vielä jäljellä. */
  public bool OhitaTyhjat(){
    while( true ){
      if( !Char.IsWhiteSpace( viim_mrk ) ){ return true; }
      int i = Console.Read();           // Luetaan merkki
      if( i == -1 ){ return false; }    // Syöte on loppunut
      viim_mrk = Convert.ToChar( i );
    }
  }

  /* Palauttaa standardisyötteen seuraavan merkkijonon.
    Jos syöte on loppunut, niin palauttaa tyhjän merkkijonon. */
  public string HaeSana(){
    if( !OhitaTyhjat() ){ return ""; }

    /* Rakennetetaan merkkijono merkki kerrallaan, kunnes nähdään
      tyhjää tai syöte loppuu. */
    StringBuilder mjono = new StringBuilder( viim_mrk.ToString() );
    while( true ){
      int i = Console.Read();
      if( i == -1 ){ viim_mrk = '\n'; return mjono.ToString(); }
      viim_mrk = Convert.ToChar( i );
      if( Char.IsWhiteSpace( viim_mrk ) ){ return mjono.ToString(); }
      mjono.Append( viim_mrk );
    }

  }

  /* Lukee syötteestä tulevan kokonaisluvun.
    Jos ei löydy, niin heittää poikkeuksen. */
  public int HaeLuku(){ return int.Parse( HaeSana() ); }

}
```

Kuva 6: C# luokka lukujen ja merkkijonojen lukemiseksi

```
// Tähän luokka kuvasta 6

class Ohjelma{
  static void Main(){
    Lukija syote = new Lukija();
      while( true ){
      int luku = syote.HaeLuku();
      if( luku == 0 ){ break; }
      string merkkijono = syote.HaeSana();
      Console.WriteLine( luku + " " + merkkijono );
    }
  }
}
```

Kuva 7: C# esimerkki lukujen ja merkkijonojen lukemisesta

lukea luvun ja syötteessä tulee merkkijono silloin kun se haluaa lukea merkkijonon. Ohjelmia käytettäessä ja ohjelmointivirheitä etsittäessä on kuitenkin usein hyödyllistä, jos ohjelma kertoo, että se ei saanut sellaista syötettä kuin oletti saavansa. Merkkijonoille tämä voidaan toteuttaa lisäämällä lukemisen perään

```
if( merkkijono == "" ){
  Console.WriteLine( "Merkkijono puuttuu!" ); break;
}
```

Kokonaisluvuilla saman saa lukemalla luku merkkijonona ja muuttamalla se luvuksi toiminnolla, joka palauttaa `false` , jos merkkijono ei ole kelvollinen luku:

```
if( !int.TryParse( syote.HaeSana(), out luku ) ){
  Console.WriteLine( "Luku puuttuu!" ); break;
}
```

Tällaisten lisäysten tekeminen on vapaaehtoista. Niitä ei vaadita kurssin ohjelmointitehtävissä.

Osassa kurssin ohjelmointitehtäviä syöte kannattaa lukea rivi kerrallaan. Rivin voi lukea toiminnolla `Console.ReadLine()` . Se lukee myös rivin lopussa olevan rivinsiirron `'\n'` , mutta ei sisällytä sitä palautettuun merkkijonoon.

Syötteen loppumisen voi tunnistaa yrittämällä lukemistoimintoa ja testaamalla sen jälkeen, onnistuiko lukeminen eli onko saatu rivi jotain muuta kuin `null` . Kuvassa 8 on ohjelma, joka kopioi kaiken syötteen tulosteeksi siten, että kunkin tyhjän rivin sijaan se tulostaa rivin, jossa lukee `"Tyhjä rivi"` . Tässä yhteydessä tyhjä rivi tarkoittaa riviä, jolla on vain välilyöntejä tai ei edes niitäkään.

```
using System;
while( true ){
  string rivi = Console.ReadLine();
  if( rivi == null ){ break; }
  bool tyhja = true;
  for( int i = 0; tyhja && i < rivi.Length; ++i ){
    tyhja &= rivi[i] == ' ';
  }
  if( tyhja ){ Console.WriteLine( "Tyhjä rivi" ); }
  else{ Console.WriteLine( rivi ); }
}
```

Kuva 8: C# esimerkki riveittäin lukemisesta