OHJELMOINNIN OPETUSMATERI-AALIN MODERNISOINTI

Juuso Hauvala

Opinnäytetyö
Kesäkuu 2014
Tietotekniikka
Ohjelmistotekniikka

TAMPEREEN AMMATTIKORKEAKOULU
Tampere University of Applied Sciences

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietotekniikan koulutusohjelma
Ohjelmistotekniikan suuntautumisvaihtoehto

HAUVALA, JUUSO:
Ohjelmoinnin opetusmateriaalin modernisointi

Opinnäytetyö 59 sivua, joista liitteitä 13 sivua
Kesäkuu 2014

Opinnäytetyön aiheena on uuden version toteutus selaimella käytettävästä C++- ja Java-ohjelmointikielten opintomateriaalista. Materiaalin alkuperäinen versio toimi vain, jos selaimelle oli asennettuna Adobe Shockwave Player. Uusi versio on tarkoitus toteuttaa HTML5:n avulla ja siten tehdä siitä toimiva käyttäen pelkästään webin omia tekniikoita.

Materiaali perustuu simulaatioihin, jotka jakautuvat esimerkki- ja harjoitussimulaatioihin. Esimerkkisimulaatiot näyttävät vaiheelta miten C++:lla ja Javalla kirjoitetut lähdekoodit toimisivat, jos ne olisivat toteutettu oikeina sovelluksina. Simulaatiot näyttävät mitä ohjelma tulostaisi näytölle. Lisäksi ne näyttävät miten muuttujat ja niiden arvot sekä tietorakenteet tallentuvat muistiin, sekä miten valinta- ja toistolauseissa olevia ehtolauseita käsitellään. Harjoitussimulaatioissa käyttäjän pitää syöttää muuttujille tiettyjä arvoja, jotta ne toteuttaisivat tehtävänannon määrittämät ehdot.

Simulaatiot käsittelevät ohjelmoinnista seuraavia aihealueita: muuttujat, valinta- ja toistorakenteet, taulukot, funktiot, osoittimet, tietueet, tiedostonkäsittely, rekursio ja olio-ohjelmointi.

Asiasanat: modernisointi, opetusmateriaali, ohjelmointi

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in Information Technology
Option of Software Programming

HAUVALA, JUUSO:
Programming teaching materials modernization

Bachelor's thesis 59 pages, appendices 13 pages
June 2014

Subject of the thesis is creating a new version of a teaching material of C++ and Java able to be run through a web browser. The original version of the material was only able to run with Adobe Shockwave Player installed in web-browser. A new version was supposed to be developed by using HTML5 when it would use only the network's own techniques.

The teaching material is based on simulations that are separated into example simulations and practice simulations. Examples that show stage to stage how source codes written in C++ and Java would work if they would be made as real applications. Simulations show what programs would write on screen. They would also show how variables and data structures would be saved in computer's memory, and how conditional statements or loop statements would be checked. In practices, the user must enter the variables certain values in order to meet the specified conditions.

The simulations of the teaching material are about the subjects of programming: variables, conditional and loop statements, arrays, functions, pointers, structures, input/output file streams, recursion and object-oriented programming.

Key words: modernization, teaching materials, programming

SISÄLLYS

LYHENTEET JA TERMIT

TAMK	Tampereen ammattikorkeakoulu
sovelma	epäitsenäinen, Web-selaimen yhteydessä suoritettava Java-ohjelma
API	ohjelmointirajapinta
C++	C-kielestä kehitetty ohjelmointikieli, joka julkaistiin vuonna 1983
Java	tulkattava oliopohjainen ohjelmointikieli, joka julkaistiin vuonna 1995

1 JOHDANTO

Tämän opinnäytetyön tarkoituksena on kuvata tekniikka ja tapa, jolla ohjelmointikielten C++ ja Javan perusasioiden opetukseen tarkoitettu HTML-opintomateriaali on päivitetty toimimaan HTML5 standardin mukaisesti. Työn on tilannut TAMKin opettaja Esa Kujansuu.

Opetusmateriaalin toimintaperiaatteena on HTML-tiedostoihin liitetyt simulaatiot, jotka jakautuvat kahteen tyyppiin: esimerkit ja harjoitukset. Esimerkit havainnollistavat esimerkillisesti ohjelmakoodeja vaihe vaiheelta näyttäen miten ne toimisivat oikeina sovelluksina. Harjoituksissa käyttäjän on annettava alustamattomille muuttujille oikeat arvot, jotta tehtävän antamat ehdot täyttyvät oikein.

Alun perin kukin simulaatio oli toteutettu täysin erillisenä, mutta tilaajan toiveesta uudessa versiossa on pyritty luomaan kaikille simulaatioille yhteinen tiedosto, joka määrittää päätoiminnallisuuden. Lisäksi päivitetyn version olisi tarkoitus olla yhteensopivampi nykyisten selainten kanssa, sillä osaa niistä ei ollut käytössä, kun alkuperäinen opetusmateriaali toteutettiin.

2 TOTEUTUSYMPÄRISTÖ JA VÄLINEET

2.1. Alkuperäisversion tutkiminen ja testaaminen

Uuden version toteuttamiseksi oli ensin tutkittava ja testattava alkuperäisiä simulaatioita, jotka oli toteutettu Macromedia Director 8.5 Educational Versionilla, joka julkaistiin jo vuonna 2001.

Ohjelman nimi on muuttunut useaan otteeseen sen ensimmäisten versioiden ilmestymisestä. Ensimmäinen versio, VideoWorks, julkaistiin vuonna 1985. Sen nimi muutettiin Macromedia Directoriksi vuonna 1994, kun julkaistiin ohjelman neljäs versio. Vuoden 2004 jälkeen ohjelma myytiin Adobelle ja viimeisin julkaistu versio on Adobe Director 12 vuoden 2013 helmikuulta (Adobe Director).

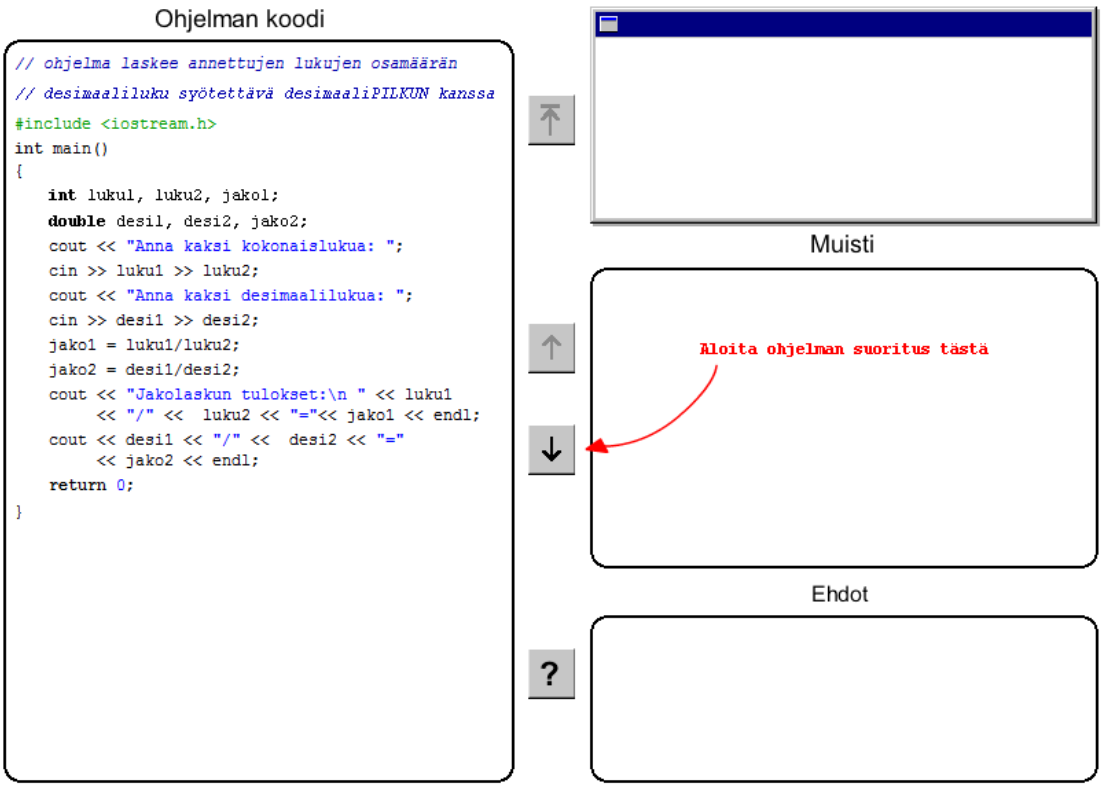
Adobe Directorilla tehtyjä animaatioita tai interaktiivisia sovelluksia on mahdollista tutkia selaimella vain, jos Adobe Shockwave Player -lisäosa on asennettu. Se on ladattavissa ilmaiseksi Adoben nettisivuilta.

Kuvan 1 simulaatiossa olevista painikkeista se, mihin punainen nuoli osoittaa paitsi aloittaa simulaation läpikäymisen, myös siirtää simulaatiossa yhden vaiheen eteenpäin. Sitä ylempi painike siirtää yhden vaiheen verran taaksepäin ja ylin taas palauttaa koko simulaation takaisin alkuvaiheeseen. Uuden version simulaatioiden olisi tarkoitus toimia lähes samankaltaisella periaatteella. Vanhoissa simulaatioissa alimmalla painikkeella ei ollut lainkaan toiminnallisuutta, joten sille uusissa simulaatioissa tarvitsisi luoda vastinetta.

2.2. Uuden version työkalut

Alkuperäinen materiaali koostui vain HTML-dokumenteista, jotka itse eivät kirjoita tai näytä mitään omaan HTML-koodiinsa kirjoitettua, vaan näyttävät sisältöä, joka on määräilty erillisissä DIR- ja DCR-tiedostoissa. Mutta uudistettu materiaali on toteutettu puhtaasti HTML5:n mukaisesti. Yleisesti sillä voidaan tarkoittaa sovellusten toteutusta webin avoimilla tekniikoilla kuten hyödyntämällä HTML-dokumenttien kautta toimivissa

web-sovelluksissa ohjelmointikielenä JavaScriptiä sekä tyylikieltä nimeltä CSS. Näillä kielillä toimivat dokumentit olivat kirjoitettavissa Notepad++ -editorilla. Uusissa simulaatioissa kaiken toiminnallisuuden on tarkoitus olla kirjoitettuna pääosin suoraan HTML-dokumentteihin.



KUVA 1. Jakolaskuja laskeva simulaatio vanhassa materiaalissa.

2.2.1 HTML

HTML (*Hypertext Markup Language* eli *hypertekstin merkintäkieli*) on tunnetuin avoimesti standardisoitu kuvauskieli. Sitä käytetään kuvaamaan hyperlinkkejä sisältävää tekstiä ja sillä voidaan myös merkitä tekstin rakennetta, kuten määrittämään otsikko ja leipäteksti. Nykyään se tunnetaan parhaiten kielenä, jota käytetään nettisivujen muodostamiseen käytettävien dokumenttien koodaamiseen.

Ensimmäinen kuvaus HTML:stä julkaistiin vuoden 1991 lopulla. Se sisälsi 22 elementtityyppiä, joista nykyään on käytössä vain 13. Vuonna 1994 kansainvälinen yritysten ja yhteisöjen yhteenliittymä W3C *(World Wide Web Consortium*) alkoi ylläpitää HTML-standardia.

HTML:n viimeisimmät vakaat versiot 4.0 ja 4.1 standardisoitiin jo vuosina 1997–1999. Uusimman version, HTML5:n, kehitys aloitettiin vuonna 2004. Sen kerrottiin alun perin valmistuvan vasta vuonna 2022, mutta nyt sen vakaan version on ilmoitettu valmistuvan vuoden 2014 loppuun mennessä. Osa sen ominaisuuksista on kuitenkin jo käytössä. Version HTML 5.1 odotetaan valmistuvan 2016. (W3C HTML Working Group) (Plan 2014)

2.2.2 JavaScript

JavaScript on Web-ympäristössä käytettävä skriptikieli eli komentosarjakieli. Sen tärkein omaisuus on lisätä dynaamista toimintaa nettisivuille sulautettuna HTML:ään. Nimestään huolimatta kieltä ei tule sekoittaa Javaan. Javan kehityksestä vastaa Sun Microsystems ja JavaScriptin kehityksestä Netscape Communications Corporation. Eikä näillä kahdella kielellä juuri muutenkaan mitään yhteistä. (JavaScript Overview – JavaScript)

Kielen viimeisin vakaa versio on 1.8.5 ja versiosta 1.3.0 alkaen JavaScriptin standardit ovat pohjautuneet EcmaScript-standardiin ECMA-262. (New in JavaScript 1.8.5 - JavaScript | MDN)

ESIMERKKI 1. JavaScriptin sulautus HTML-koodiin

```
<html>
 <head>
  <title>JavaScript Example</title>
 </head>
 <body>
  <script>
   document.write("Hello world!");
  </script>
 </body>
</html>
```

JavaScriptiä voidaan sulauttaa HTML:ään kirjoittamalla koodia `<script>`-elementtien sisään. Esimerkin 1 mukainen HTML-dokumentti tulostaisi JavaScriptin avulla tekstin *Hello world!*

2.2.3 CSS

CSS (*Cascading Style Sheets* eli *porrastetut tyyliarkit***)** on tyylikieli ja kehitetty erityisesti WWW-dokumenteille.

CSS:n viimeisin vakaa versio on CSS3. Siihen sisältyy elementtien, laatikoiden ja fonttien muuntaminen sekä ominaisuuksia, jotka on suunniteltu osittain korvaamaan Java-sovelmat sekä Flash-animaatiot. (Cascading Style Sheets)

2.2.4 Notepad++

Notepad++ on Windowsille suunnattu ilmainen teksti- ja lähdekoodieditori, jonka viimeisin versio 6.6.2 julkaistiin 8. toukokuuta 2014. Se tukee useiden ohjelmointikielten syntaksin korostusta. Se on kirjoitettu C++:lla ja lisäksi hyödyntää vapaata Scintilla-komponenttia sekä käyttää puhdasta Win32-APIa sekä STL:ää (*Standard Template Library*), mikä varmistaa editorille paremman toimintanopeuden ja sillä kirjoitetuille dokumenteille pienemmän tiedostokoon. (Notepad++ Home)

3 TOTEUTUSSUUNNITELMA

3.1. Alkutarkastukset

Aivan ensimmäiseksi oli tarkoitus tarkastella, miten vanhan materiaalin simulaatiot toimivat. Oli selvitettävä, millaisiin vaiheisiin kunkin simulaation käsittely jaetaan, missä järjestyksessä ne käydään läpi sekä mitä niissä tapahtuu.

Sen jälkeen tarkistettiin, että löytyikö alkuperäisen materiaalin simulaatioiden joukossa sellaisia, jotka eivät olleet toimivia. Tällaiset simulaatiot oli sovittu poistettavaksi päivitetystä versiosta kokonaan. Nämä yksityiskohdat oli päivitettävä myös niihin HTML-tiedostoihin, joihin tehdyistä valikoista käyttäjä voi siirtyä tarkastelemaan haluamiaan simulaatioesimerkkejä tai harjoituksia.

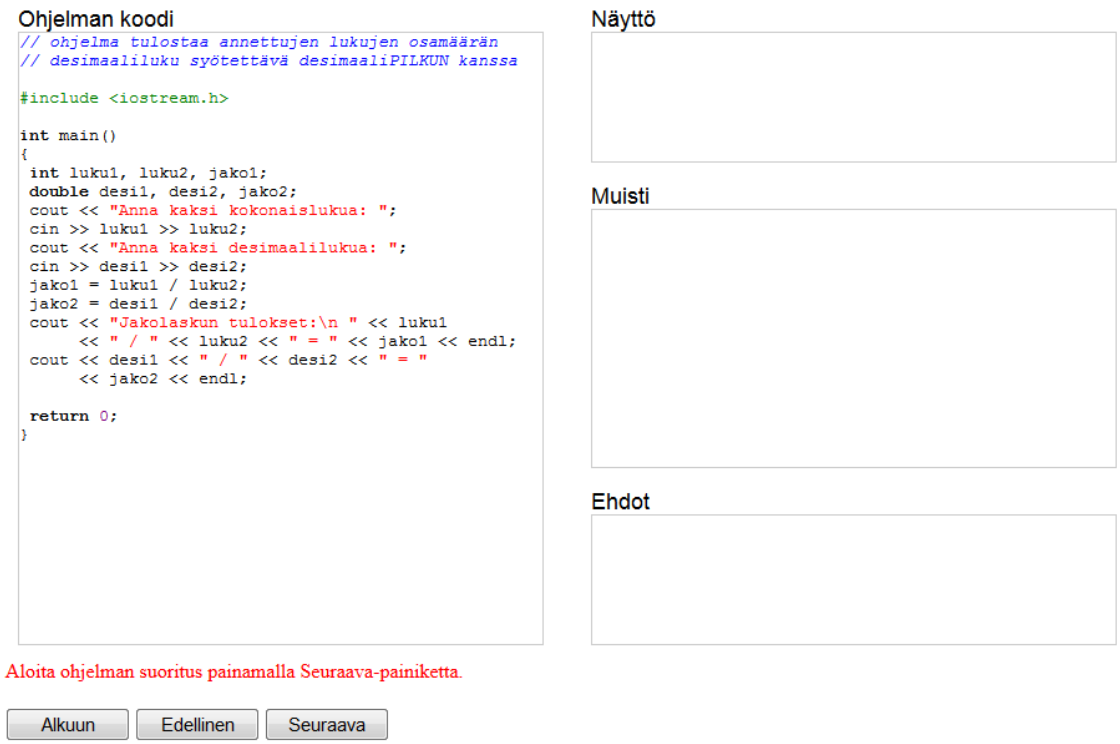
Joistain valikoista puuttui linkkejä sellaisiin simulaatioihin, jotka toimivat, ja tämä yksityiskohta oli korjattava myös.

3.2. Uusien tiedostojen luominen

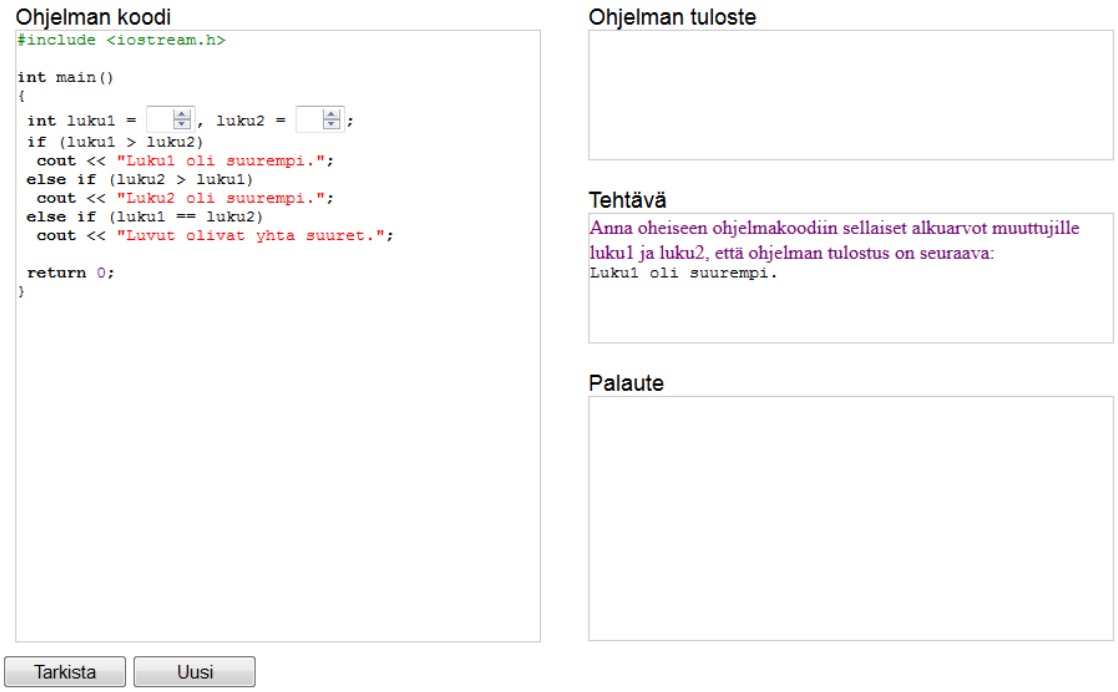
Kustakin havainnollistavasta esimerkistä tai harjoituksesta oli tehtävä uudet tiedostot. Alkuperäiset simulaatiot oli liitetty HTML-dokumentteihin DCR- ja DIR-tyyppisten tiedostojen kautta. Mutta uusissa simulaatioiden näkymä on määritelty vain HTML-dokumentin sisällön mukaan. Kopioi-liitä -menetelmä on käytettävissä ainoastaan tekstiin, joka oli dokumentoidussa muodossa. Ja koska alkuperäisen materiaalin simulaatioissa käsiteltävät koodit olivat osa Shockwave Playerilla ajettavaa sovellusta, uudet tiedostot täytyi kirjoittaa käsin.

Aluksi pyrittiin kirjoittamaan uusiin HTML-tiedostoihin alkuperäisten simulaatioiden esimerkkikoodien teksti. Korostaakseen siinä olevien tiettyjen sanojen tai merkintöjen tarkoitusta, oikeat ohjelmointityökalut näyttävät niitä tietyillä väreillä tai käsittelevät muuten tekstin graafista muotoa. Siitä syystä HTML-dokumentin käyttäjälle näytettävä teksti oli myös muotoiltava siihen sopivaksi.

Lisäksi oli luotava kuhunkin simulaatioon näkymä, jossa näytetään kolme tai neljä eri ikkunaa. Kuvassa 2 ensimmäinen ikkuna näyttäisi itse käsiteltävän ohjelmakoodin. Toiseen ikkunaan tulostuisi tekstiä samalla tavalla kuin oikeassa C++- tai Java-ohjelmassa aina läpikäydyn vaiheen mukaisesti. Esimerkkisimulaatioissa kaksi muuta ikkunaa näyttävät muistiin tallennetut muuttujat sekä ehtolauseiden käsittelyssä sen ovatko ne tosia vai epätosia. Ehtolauseiden käsittelyä näyttävää ikkunaa ei ole kuitenkaan kaikissa esimerkkisimulaatioissa. Kuvan 3 mukaisessa harjoitussimulaatioissa viimeiset kaksi ikkunaa näyttivät tehtävänannon sekä palautteen siitä oliko käyttäjä antanut oikeat arvot tyhjiin kenttiin.



KUVA 2. Jakolaskuja tekevän C++-simulaation aloitustilanne



KUVA 3. if-else if -rakenteita havainnollistavan harjoitussimulaation aloitustilanne

Kuvassa 2 nähtävän HTML-dokumentin lähdekoodi on katsottavissa liitteestä 1 ja kuvan 3 dokumentin lähdekoodi liitteestä 2. Luvussa 4 on selostettuna simulaatioiden toiminnallisuus.

3.3. Simulaatiopohjan luominen

Uutta rakennetta varten luotiin kaksi tiedostoa: simulaatio.js ja sheets.css. Näillä tiedostoilla määritellään pohja kaikkien simulaatioiden yhtenäisille ominaisuuksille. Toiseen näistä tiedostoista sisältyy JavaScript-funktioita, jotka määrittävät esimerkkisimulaatioissa vaiheesta toiseen siirtymisen sekä harjoitussimulaatioissa oikeiden vastausten tarkastelun. Toisessa tiedostossa määritellään CSS:ää käyttäen simulaatioiden painikkeiden sekä syötekenttien ulkomuodot. Molemmat tiedostot ovat koko materiaalissa kahtena kappaleena. Toiset ovat samassa kansiossa C++-simulaatioiden kanssa ja toiset samassa kansiossa Java-simulaatioiden kanssa.

3.4. Simulaatioiden viimeistelyt

Lopuksi oli tarkoitus toteuttaa erikseen kullekin simulaatiolle tämän oma toiminnallisuus eli se mitä sen on tarkoitus tulostaa näytölle sekä miten se käsitellään kunkin vaiheen. Kukin simulaatio käsittelee ohjelmointikielen toimintaa eri aihealueita. Luvussa 4 selitetään tarkemmin miten tämä osuus koko työstä on toteutettu eri simulaatiotyypeissä.

4 TOIMINNALLISUUS

4.1. Esimerkkisimulaatiot

4.1.1 Päätoiminnallisuus

Tavallisten ohjelmaesimerkkien simulaatiot ovat sellaisia, joita voidaan käydä läpi vaihe kerrallaan. Ohjelmoinnissa voidaan lukea eri vaiheiksi seuraavat yksityiskohdat:

- ohjelmakirjaston ja siinä olevien luokkien ja metodien sisällyttäminen
- muuttujan luominen/alustaminen (mahdollista toteuttaa molemmat samassa vaiheessa)
- tekstin tulostaminen
- syötteen antaminen
- syötteen arvon tallentaminen muuttujaan
- valinta- tai toistorakenteen tyypin määrittäminen
- rakenteessa olevan ehtolauseen tarkistaminen
- arvon palauttaminen funktion tai pääohjelman lopussa

Esimerkkisimulaatioissa on kolme painiketta: *Alkuun*, *Edellinen* ja *Seuraava*. Niiden toimintaperiaate vastaa painikkeiden nimiä. *Alkuun* palauttaa koodin läpikäymisen takaisin ensimmäiseen vaiheeseen, *Edellinen* palauttaa yhden vaiheen taaksepäin ja *Seuraava* siirtää yhden vaiheen eteenpäin. Kuten lähdekoodipohjasta näkyy, kukin painike kutsuu samaa funktiota, mutta antaa muuttujalle toiminto eri arvon.

Ennen vaiheesta toiseen siirtymistä kukin simulaatio tarkistaa dokumenttiin kirjoitetun JavaScript-koodinsa kautta onko käyttäjä tehnyt kaiken tarvittavan, jotta seuraavaan vaiheeseen voidaan siirtyä. Jos esteitä ei ole, siirrytään simulaatiopohjatiedoston simulaatio.js avulla seuraavaan vaiheeseen. Sen jälkeen vielä dokumenttiin kirjoitettu koodi vielä muuttaa simulaation näkymän edeltävän tai seuraavan vaiheen mukaiseksi. Painikkeesta ja sitä kautta muuttujan toiminto arvosta kutsutaan jotakin tiedoston simulaatio.js funktioista, joihin välitetyt muuttujat määrittävät mihin vaiheeseen simulaatio siirtyy.

Se, että mihin vaiheeseen painikkeitta painettaessa siirrytään, määräytyy funktion paivita() alussa olevista ehdoista. Jos simulaatiossa ei käsitellä ehto- tai toistolauseita, kirjoitetaan vain yksinkertaisesti esimerkissä 2 nähtävällä tavalla.

ESIMERKKI 2. Oletusarvon määrittely muuttujille

```
edvaih = vaihe - 1;
seurvaih = vaihe + 1;
```

Muuttuja edvaih välitetään tiedoston simulaatio.js funktioon edvaihe(), jos halutaan palata edeltävään vaiheeseen. Muuttuja seurvaih taas välitetään funktioon seurvaihe(), jos halutaan edetä seuraavaan vaiheeseen. Näitä funktiota kutsutaan myöhemmässä osassa ohjelmakoodia. Poikkeustapaukset näiden kahden muuttujan arvoista valinta- ja toistorakenteita käytettäessä ovat tarkemmin selostettu luvussa 4.1.3.

Jotta käyttäjä näkee mitä käsiteltävän ohjelmakoodin (kirjoitettuna HTML-dokumentissa div-osion ”koodi” sisäpuolelle) kussakin vaiheessa käsitellään, kyseinen osa ohjelmakoodista on merkittynä oranssinsävyisellä värillä FF6600. Samalla värillä on myös merkittynä se muuttuuko jokin ohjelman luoma muuttujan arvoista.

Käsiteltävän vaiheen vaihemerkintä on tehty mahdolliseksi siten, että kukin ohjelman vaiheista on jaettuna dokumentin HTML-koodissa `<span>`- tai `<div>`-elementteihin, joita on kussakin dokumentissa käytetty sen mukaan sijoittuvatko jotkin osat ohjelmakoodista samalle vai eri riville. Jokaisen elementin id-osioksi on merkitty numero ja tähän numeroon reagoimalla voidaan siirtää tai lisätä värimerkintä. Esimerkiksi, jos aloitustilanteesta halutaan päästä ensimmäiseen vaiheeseen. Osion `<div id=”1”>` tai `<span id=”1”>` eteen lisätään elementti `<span style='color: FF6600;'>`. Mutta samalla tallennetaan myös yhteen simulaatio.js -tiedoston muuttujista div-osion tai span-osion alkuperäinen sisältö, jolloin vaiheesta toiseen siirryttäessä edellisen vaiheen teksti palaa alkuperäiseen väriinsä. Tämän mahdollistamiseksi jokaisessa simulaatiossa funktiossa paivita() on heti muuttujien edvaih ja seurvaih arvojen määrittämisen jälkeen esimerkin 3 if-lause, jossa muutetaan muuttujan vaihe arvo merkkijonomuotoon ja tallennetaan samannimisen span- tai div-osion sisällöksi muuttujan alkuperSisalto sisältö, jossa on sama teksti, mutta ilman värillä FF6600 toteutettua värjäystä. Kun värimerkinnän siirtyminen on tapahtunut, voidaan siirtyä esimerkin 4 mukaiseen if-else if -rakenteeseen.

ESIMERKKI 3. Värimerkinnän poistaminen viimeksi käsitellystä vaiheesta

```
if (vaihe != 0) {
   // Seuraava if-else -rakenne varmistaa
   // että oranssia värimerkintää
   // vailla oleva versio tekstistä säilyy,
   // vaikka ohjelma pysähtyisi
   // syötteiden kanssa ilmenemien ongelmien vuoksi.
   var luku = vaihe.toString();
   document.getElementById(luku).innerHTML =
   alkuperSisalto;
   }
```

ESIMERKKI 4. Oletusmäärittely tiedoston simulaatio.js funktioiden kutsumiselle

```
if (toiminto == 0) alkuun();
else if (toiminto == 1) edvaihe(edvaih);
else if (toiminto == 2) seurvaihe(seurvaih);
```

Kyseinen if-rakenne olisi kirjoitettuna tuohon muotoon vain simulaatiossa, jossa ei ole valinta- tai toistorakenteita tai ainuttakaan vaihetta, jossa tarvitsisi antaa muuttujille arvoja. Kuten aiemmin kerrottu, muuttujan toiminto arvosta riippuu mitä funktiota kutsutaan. Nämä funktiot löytyvät vain tiedostosta simulaatio.js ja siellä ne ovat määritetty esimerkissä 5 nähtävällä tavalla.

ESIMERKKI 5. Tiedoston simulaatio.js funktioita

```
function alkuun() {
    if (vaihe > 1) vaihe = 1;
}
function edvaihe(edvaih) {
    if (vaihe > 1) vaihe = edvaih;
}
function seurvaihe(seurvaih) {
    if (vaihe < max) vaihe = seurvaih;
}
```

Värimerkinnän poistamisen ja muuttujan vaihe arvon muuttamisen jälkeen vielä lisättiin värimerkintä uuden vaiheen kohdalle kutsumalla esimerkin 6 kohdalla funktiota merkinta().

ESIMERKKI 6. Funktion merkinta() kutsuminen

```
// Tarkistetaan funktion kautta
voidaanko värimerkintä siirtää.
merkinta();
```

Kussakin HTML-dokumentissa kutsuttava funktio merkinta() on sisällytettynä funktioiden alkuun(), edvaihe() ja seurvaihe() tavoin vain tiedostossa simulaatio.js. Se varmistaa, että vaihe, johon käyttäjä on juuri siirtynyt esimerkkisimulaatiota ajaessaan, muuttuu toisenväriseksi ja siten osoittaa käyttäjälle kyseisen kohdan olevan se vaihe, mitä simulaatiossa parhaillaan käsitellään. Esimerkissä 7 nähdään miten funktio toimii.

ESIMERKKI 7. Funktion merkinta() toiminnallisuus

```
function merkinta() {
    if (esto == false) {
       var luku = vaihe.toString();
       alkuperSisalto =
       document.getElementById(luku).innerHTML;
       document.getElementById(luku).innerHTML =
       "<span style='color: FF6600;'>" +
       document.getElementById(luku).innerHTML;
       // Siirtää värimerkinnän esimerkkikoodin vaiheesta
       // toiseen vaiheeseen, jos sitä ei ole estetty.
       }
}
```

Kussakin esimerkkisimulaatiossa viimeisenä vaiheena, jonka jälkeen pääsee enää vain taaksepäin tai aloittamaan koko simulaation läpikäynnin alusta, on C++-simulaatioiden tapauksessa *return 0* eli arvon nolla palauttaminen sekä Java-simulaatioiden tapauksessa pääluokan pääfunktion päättyminen.

4.1.2 Syötteen antaminen

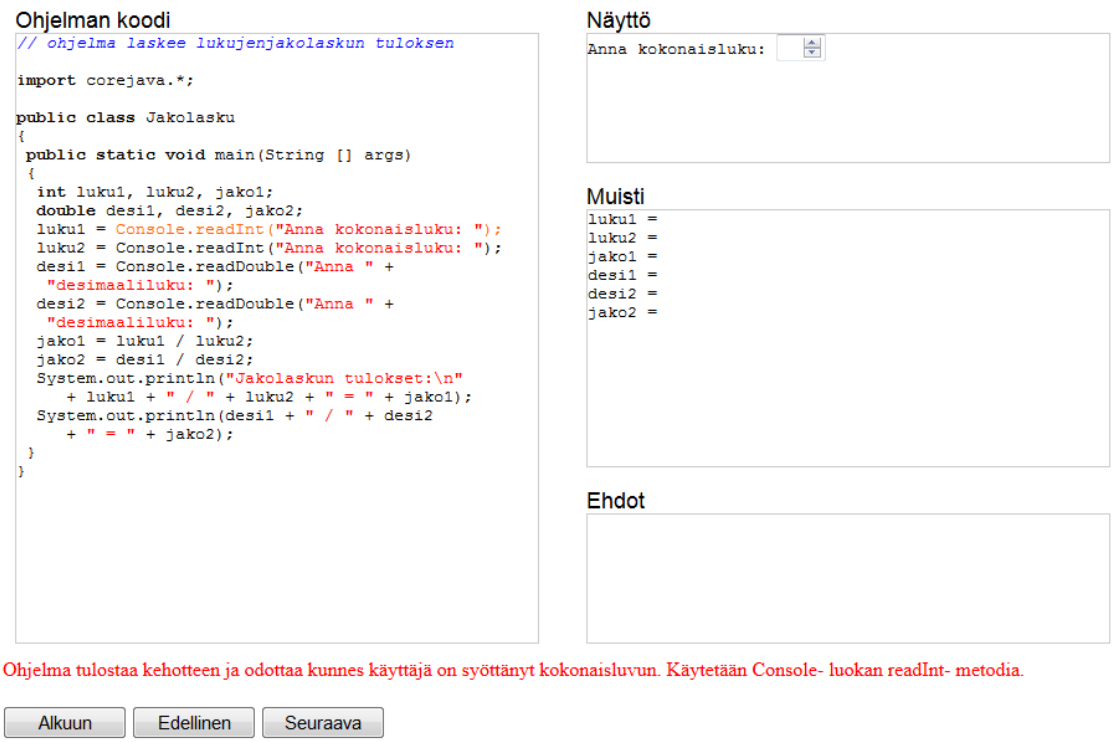
HTML-dokumenteissa olevissa syötekentissä voidaan antaa joko numeroarvoja tai tekstiä. Mutta simulaatioissa syötteet ovat rajoitettu niin, että jos oikea C++- tai Java-ohjelma vaatii sitä, se voi olla vain int (kokonaisluku), float (desimaaliluku), double (reaaliluku), char (merkki) tai string (merkkijono). Tämä vaihe on se, minkä eteneminen riippuu itse käyttäjästä.

Jos alkuperäisessä materiaalissa käyttäjä antaa int-muuttujalle arvoksi desimaaliluvun, siirtyminen toiseen vaiheeseen estetään ja syy tähän ilmoitetaan käyttäjälle JavaScriptin alert-funktiota käyttäen. Mutta oikea C++-ohjelma vain muuttaa syötetyn desimaaliluvun kokonaisluvuksi ja kyseisellä tavalla se toimii myös päivitetyssä materiaalissa. Mutta oikeiden C++-ohjelmien tavoin edes päivitetyt simulaatiot eivät int-, float- tai

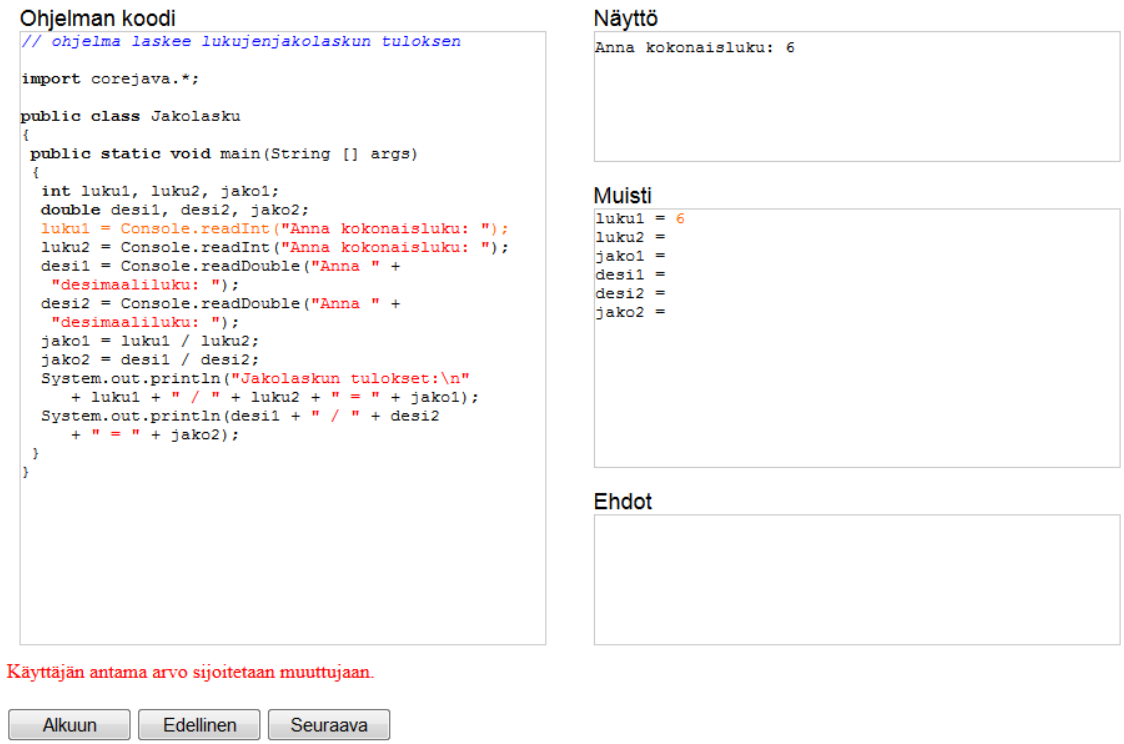
double-tyyppisten muuttujien arvoja syötettäessä hyväksy arvoja, joihin on syötetty numeroiden lisäksi muitakin merkkejä.

Riippuen syötettävien arvojen tyypeistä tai mahdollisesta suuruudesta, liitetty tiedosto sheets.css määrittää syötekenttien koon ja ulkomuodon.

Kuvan 4 tilanteesta näkyy kuinka Java-simulaatio pyytää käyttäjää antamaan syötteeksi kokonaislukuarvon ja kuvassa 5 näkyy kuinka annettu arvo on tallennettu muistiin muuttujan luku1 arvoksi.



KUVA 4. Syötteen antaminen Java-simulaatiossa

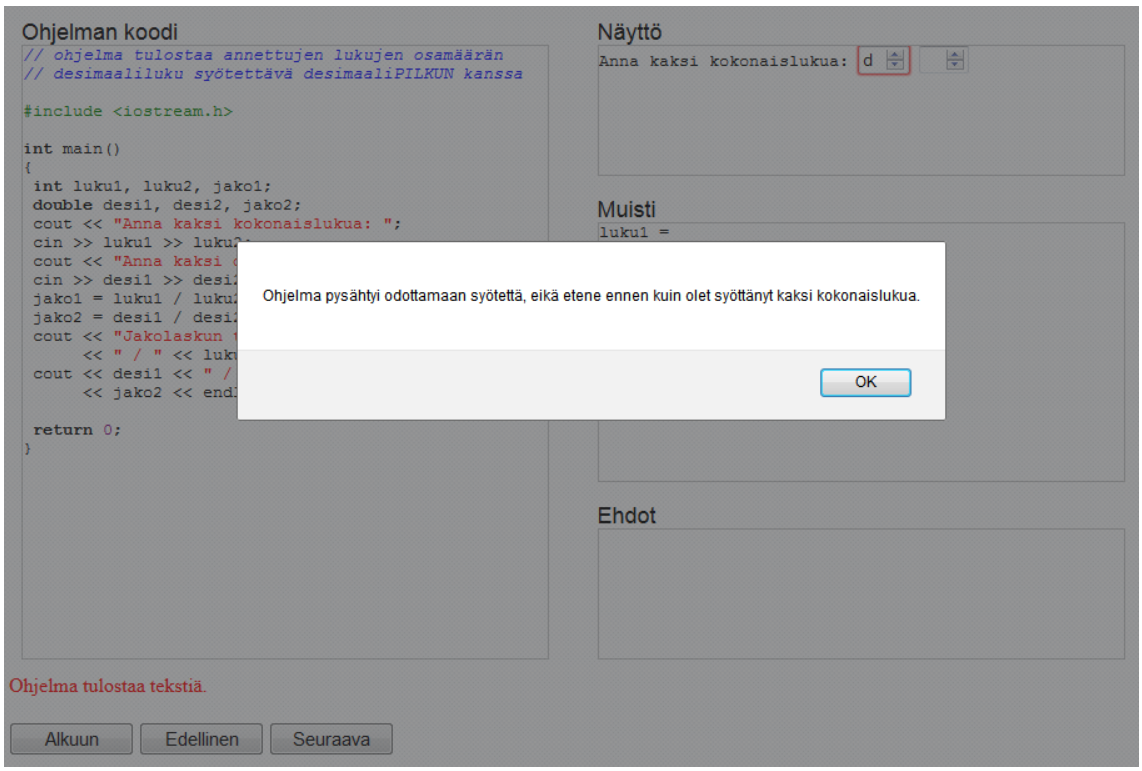


KUVA 5. Java-simulaatio syötteen antamisen jälkeen

Kuvan 6 tilanteessa käyttäjä voi antaa muuttujille luku1 ja luku2 arvot samanaikaisesti. Toisin kuin Javan readInt()-metodi, C++:n cin-operaattori sallii useamman muuttujan arvon syöttämisen yhdellä kerralla. Kuvassa 6 nähtävä simulaatio on sama kuin kuvan 2 simulaatio, joten sen lähdekoodi on nähtävissä liitteestä 1. Sen mukaan kuvan simulaatiossa on 13 vaihetta. Syötteen antaminen muuttujille luku1 ja luku2 on vaihe numero 5. Ennen syötekentän arvosta olisi JavaScriptin funktion isNaN() kautta voinut tarkistaa onko syötearvo jotain muuta kuin numero. Mutta uusimmissa selaimissa numeroita vaativat syötekentät reagoivat muihin merkkeihin samalla tavalla kuin, jos syötekenttä olisi kokonaan tyhjä. Joten siksi funktion käyttö isNaN() ei ole enää tarpeen.

KUVA 6. Syötteen antaminen C++-simulaatiossa

Kuvassa 7 nähdään, miten JavaScriptin alert-funktio kertoo syyn virheelliseen syötteeseen. Tässä tapauksessa virhe johtuu kahdesta syystä: muuttujan luku1 arvoksi on syötetty kirjain ja muuttujalle luku2 ei ole syötetty arvoa lainkaan.



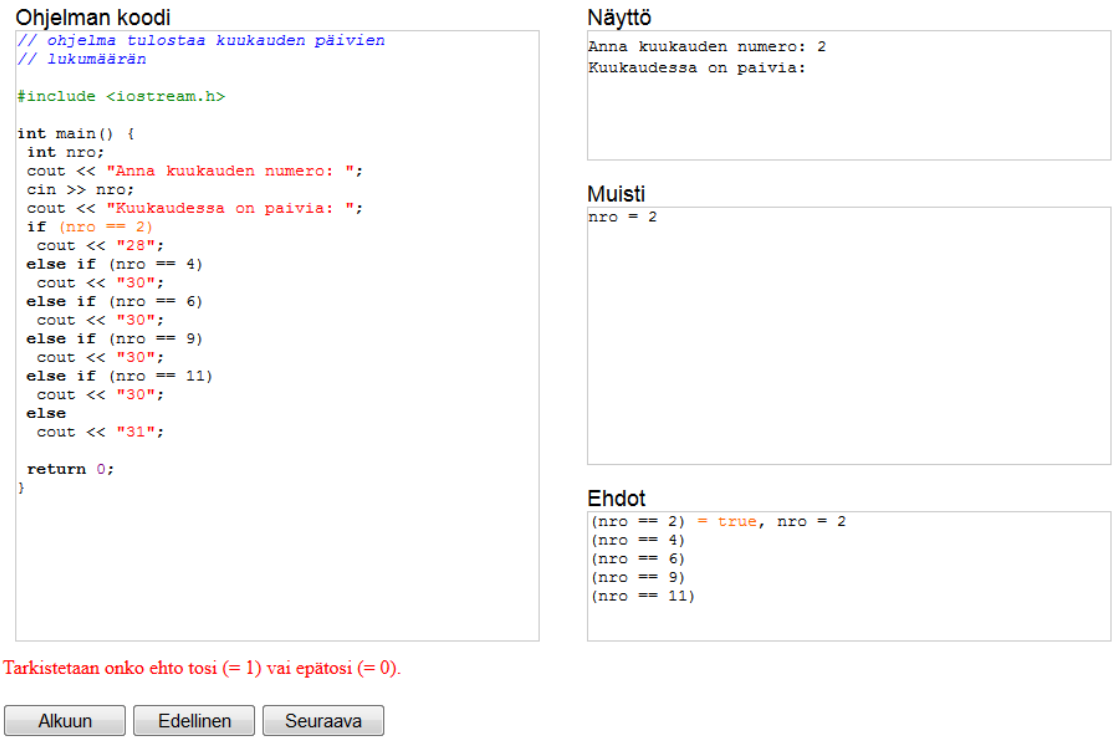
KUVA 7. Virheellinen syöte

4.1.3 Valinta- ja toistorakenteet

Valintarakenteita käsiteltäessä jotkut vaiheet voidaan joko jättää kokonaan käymättä, arvoista, jotka käyttäjä on muuttujille antanut. Valintarakenteita ovat if-rakenteet tai switch-rakenteet.

if-rakenteissa on tiettyä ohjelmakoodia, joka suoritetaan vain, jos if-lauseessa rakenteessa määritelty ehto on tosi. Jos ehto on epätosi, käsitellään else-lauseen sisältö. Joskus if-lauseen ja else-lauseen välissä voi olla yksi tai useampi else if -lause, jonka sisällön ohjelma käy läpi, jos edeltävät ehtolauseet ovat olleet epätosia. Jos yksi niistä on tosi, sitä seuraavat lauseet sekä else-lause ohitetaan kokonaan.

Koska kuvan 8 tilanteessa if-lauseen ehto on tosi, siirrytään suoraan siihen vaiheeseen, jossa tulostetaan teksti ”28”. Tämän jälkeen siirrytään suoraan kohtaan *return 0*.



KUVA 8. if-lauseen tarkistaminen C++-simulaatiossa

Esimerkissä 8 nähtävä if-else if -rakenne näyttää korvaa kuvan 8 simulaation esimerkissä 2 nähdyn määrittelytavan muuttujien edvaih ja seurvaih arvoille. Vaiheet 8, 11, 14, 17 ja 20 viittavat if- sekä else if -lauseiden sisällä oleviin ehtoihin. Vaiheet 9, 12, 15, 18 ja 21 viittavat ehtojen jälkeisiin cout-operaatioihin. Vaiheet 10, 13, 16, 19 ja 22 viittaavat if-lauseen jälkeisten ehtolauseen tyyppien, else if ja else, määrityksiin. Ja vaihe 24 viittaa simulaation loppuun eli *return 0*.

ESIMERKKI 8. Kuvan 8 simulaation vastine esimerkin 2 tapaukselle

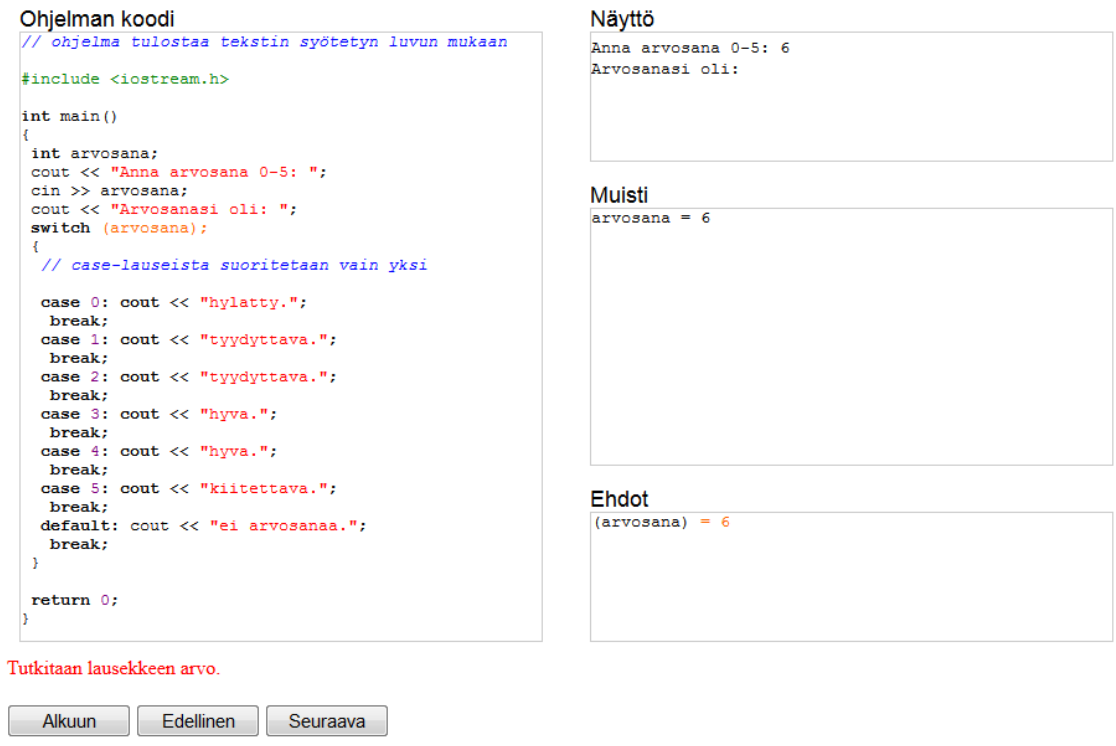
```
if (vaihe == 8 || vaihe == 11 || vaihe == 14 ||
vaihe == 17 ||
vaihe == 20) {
   edvaih = vaihe - 1;
   var nro =
   parseInt(document.getElementById('nro').value,
   10);
   if ((vaihe == 8 && nro == 2) ||
   (vaihe == 11 && nro == 4) ||
   (vaihe == 14 && nro == 6) ||
   (vaihe == 17 && nro == 9) ||
   (vaihe == 20 && nro == 11))
      seurvaih = vaihe + 1;
   else seurvaih = vaihe + 2;
   }
else if (vaihe == 9 || vaihe == 12 || vaihe == 15
|| vaihe == 18 || vaihe == 21) {
        edvaih = vaihe - 1;
        seurvaih = 24;
        }
else if (vaihe == 10 || vaihe == 13 || vaihe == 16
|| vaihe == 19 || vaihe == 22) {
        edvaih = vaihe - 2;
        seurvaih = vaihe + 1;
        }
else if (vaihe == 24) {
        var nro =
        parseInt
        (document.getElementById('nro').value, 10);
        if (nro == 2) edvaih = 9;
        else if (nro == 4) edvaih = 12;
        else if (nro == 6) edvaih = 15;
        else if (nro == 9) edvaih = 18;
        else if (nro == 11) edvaih = 21;
        else edvaih = vaihe - 1;
        seurvaih = vaihe;
        }
else {
     edvaih = vaihe - 1;
     seurvaih = vaihe + 1;
     }
```

Koodin if-lause kertoo miten kuvan 8 simulaatiossa vaiheesta toiseen siirtyminen määräytyy, jos käsitellään ehtolauseita. Ensimmäinen else if -lause käsittelee niitä vaiheita, joihin on siirrytty, jos simulaatiossa edellisen vaiheen ehto on ollut tosi. Kun rakenteen sisällä oleva ohjelmakoodi on käyty läpi, siirrytään vaiheeseen 24, mikä kuvan 8 simulaatiossa olisi *return 0* eli ohjelman lopetus. Toinen else if -lause käsittelee vaiheita, joissa on siirrytty else if -lauseisiin tai else-lauseeseen, jos if-lauseen ehto on epätosi. Ja vaiheessa 24 eli simulaation lopussa, jos halutaan palata edeltäviin vaiheisiin, muuttujan nro arvo auttaa selvittämään mikä if-else if -rakenteen lauseista oli se mitä viimeksi käsiteltiin. Jos edeltävä tai seuraava vaihe ei kuulu yhteenkään valinta- tai toistoraken-

teeseen, vaihe, johon siirrytään, on joko nykyisen vaiheen arvo vähennettynä yhdellä ja kasvatettuna yhdellä.

switch-rakenteissa tarkistetaan yhden tietyn muuttujan arvo. Jos muuttujan arvo on sama yhden case-osan vaatima arvo, siirrytään muuttujan tarkistamisesta suoraan vastaavaan case-osaan ja käydään sen sisältöä läpi, kunnes vastaan tulee *break*, minkä jälkeen switch-rakenteen läpikäynti päättyy. Jos switch-rakenteen tutkima muuttuja ei vastaa yhdenkään case-osan vaatimaa arvoa, siirrytään default-osaan ja käydään läpi sen sisältö. Jos default-osaa ei ole, switch-rakenteen läpikäynti päättyy.

Kuvan 9 tilanteessa siirrytään default-osaan, koska arvolle 6 ei ole omaa case-osaa. Sen jälkeen vain siirrytään *break*-kohtaan, jossa poistutaan switch-rakenteesta, ja tämän jälkeen suoraan kohtaan *return 0*.



KUVA 9. Muuttujan arvon tarkistaminen switch-rakenteessa

Esimerkin 9 if-rakenne HTML-dokumentin JavaScript-osiota selittää miten vaiheesta toiseen siirtymiset käsitellään kuvan 9 simulaation switch-rakenteessa. Vaiheesta toiseen siirtyminen määräytyy muuttujan arvosana arvon mukaan. Ensimmäinen else if -lause käsittelee sitä, kun ollaan siirrytty yhteen case-lauseista. Toisin kuin if-, else- ja else-lauseet, switch-lauseessa ohitetaan vlittömästi ne case-lauseet, jotka eivät vastaa

muuttujan arvosana arvoa. Toinen else if -lause käsittelee *break*-toimintoa, jonka avulla lopetetaan case-lauseen läpikäynti. Ja vaiheessa 23 eli simulaation lopussa, jos halutaan palata edeltäviin vaiheisiin, muuttujan arvosana arvo auttaa selvittämään mikä switch -rakenteen case-lauseista oli se mitä viimeksi käsiteltiin. Jos edeltävä tai seuraava vaihe ei kuulu yhteenkään valinta- tai toistorakenteeseen, vaihe, johon siirrytään, on joko nykyisen vaiheen arvo vähennettynä yhdellä ja kasvatettuna yhdellä.

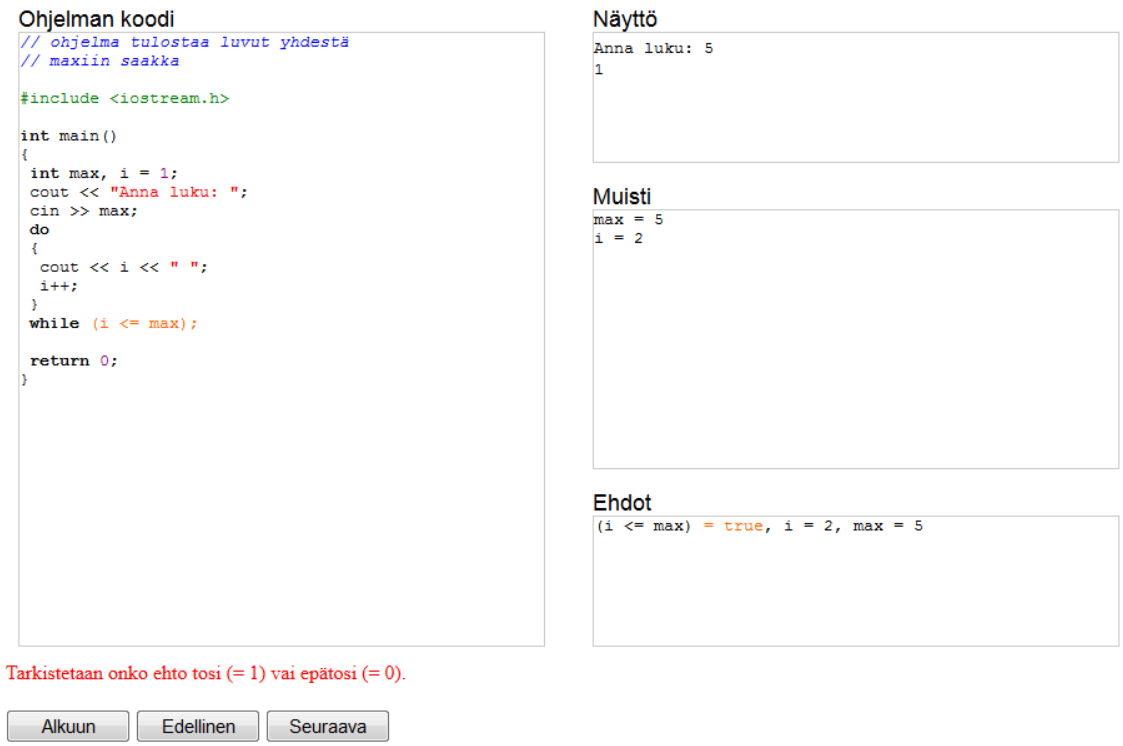
ESIMERKKI 9. Kuvan 9 simulaation vastine esimerkin 2 tapaukselle

```
if (vaihe == 8) {
   edvaih = vaihe - 1;
   var arvosana = parseInt
   (document.getElementById('arvosana').value, 10);
   if (arvosana == 0) seurvaih = vaihe + 1;
   else if (arvosana == 1) seurvaih = 11;
   else if (arvosana == 2) seurvaih = 13;
   else if (arvosana == 3) seurvaih = 15;
   else if (arvosana == 4) seurvaih = 17;
   else if (arvosana == 5) seurvaih = 19;
   else seurvaih = 21;
   }
else if (vaihe == 9 || vaihe == 11 || vaihe == 13 ||
vaihe == 15 || vaihe == 17 || vaihe == 19 || vaihe == 21) {
        edvaih = 8;
        seurvaih = vaihe + 1;
        }
else if (vaihe == 10 || vaihe == 12 || vaihe == 14 ||
vaihe == 16 || vaihe == 18 || vaihe == 20) {
        edvaih = vaihe - 1;
        seurvaih = 23;
        }
else if (vaihe == 23) {
        var arvosana = parseInt
        (document.getElementById('arvosana').value, 10);
        if (arvosana == 0) edvaih = 10;
        else if (arvosana == 1) edvaih = 12;
        else if (arvosana == 2) edvaih = 14;
        else if (arvosana == 3) edvaih = 16;
        else if (arvosana == 4) edvaih = 18;
        else if (arvosana == 5) edvaih = 20;
        else edvaih = vaihe - 1;
        seurvaih = vaihe;
        }
else {
     edvaih = vaihe - 1;
     seurvaih = vaihe + 1;
     }
```

Toistorakenteissa taas kaikki sen sisäpuolelle sijoittuvat vaiheet, käydään läpi uudestaan kunnes niille määritetty ehto on epätosi. Toistorakenteita do-while-, for- ja while-rakenteet.

do-while -rakenteessa käydään ensin läpi do-silmukan sisältö. Sen jälkeen siirrytään sen alla olevaan while-osaan ja tarkistetaan onko siellä oleva ehto tosi vai epätosi. Jos ehto on epätosi, koko rakenteen sisältö käydään uudelleen läpi.

Kuvassa 10, koska muuttujan i arvo on pienempi kuin max, jonka arvoksi on syötetty 6, silmukan ehto on epätosi. Joten käydään do-silmukka uudelleen läpi alkaen toistolauseen tyypin (tässä tapauksessa *do*) määrittelystä ja toistetaan se kunnes muuttujan i arvo on 6. Silmukan sisällä muuttujan i arvoa kasvatetaan kerran yhdellä. Ja kun se on while-osan kohdalla sama kuin muuttujan max arvo, do-while -rakenne päättyy.



KUVA 10. Ehtolauseen käsittely do-while -rakenteessa

Varmistaakseen, että edelliset vaiheet käsitellään uudestaan, jos toistolauseen ehto on epätosi, käytetään tiedostosta simulaatio.js otettua toisto-taulukkomuuttujaa. Esimerkissä 9 näkyy otteita lähdekoodista kuvan 10 simulaatioon.

Taulukon toisto alkio 0 vastaa muuttujan i arvoa. Vaihe 8 on kohta, jossa muuttujan i arvoa kasvatetaan yhdellä. Jos siirrytään simulaatiossa yksi vaihe taaksepäin, varmistetaan että muuttujan arvoa vähennetään yhdellä. Toisto-taulukon alkioita voidaan käyttää paitsi siinä tapauksessa jos joidenkin muuttujien arvo kasvaa tai vähene simulaation

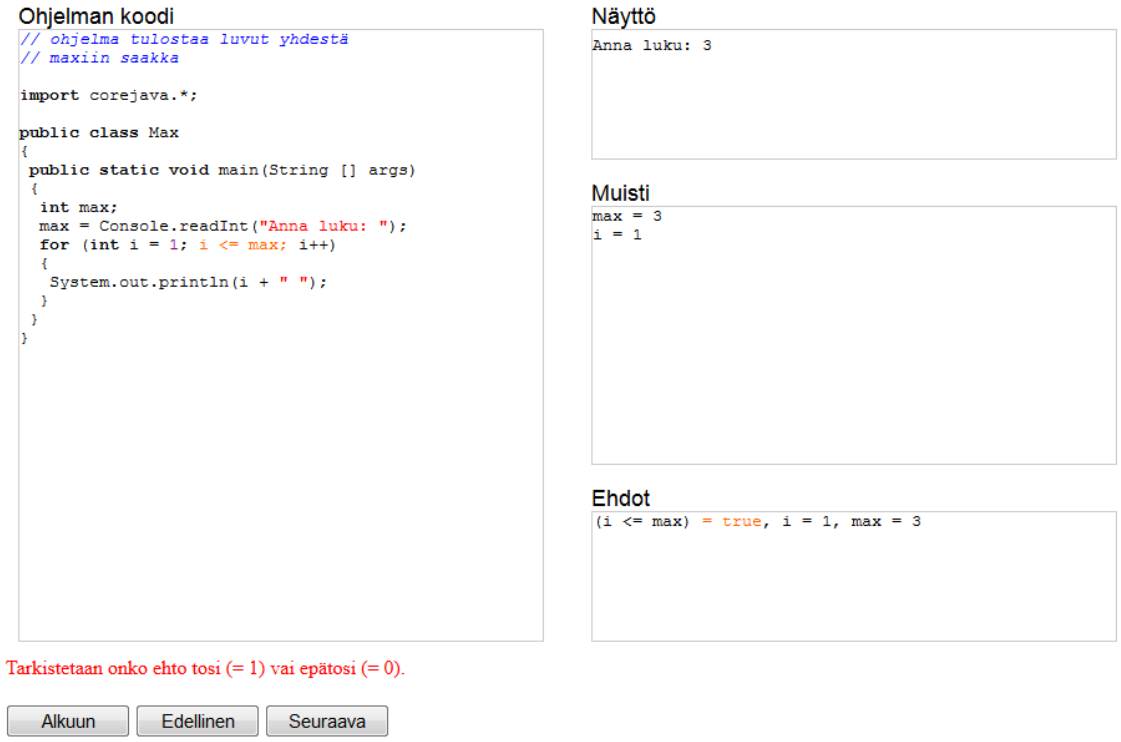
ajon aikana, ja kukin sen alkioista, joita simulaatiossa käytetään, on alustettu HTML-dokumentissa etukäteen.

ESIMERKKI 10. Kuvan 10 simulaation vastine esimerkin 2 tapaukselle

```
if (vaihe == 6) {
   if (toisto[0] > 1) edvaih = 10;
   else edvaih = vaihe - 1;
   seurvaih = vaihe + 1;
   }
else if (vaihe == 10) {
        var luku = parseInt(document.getElementById('max').value);
        edvaih = vaihe - 1;
        if (toisto[0] <= luku) seurvaih = 6;
        else seurvaih = vaihe + 1;
        }
else {
     edvaih = vaihe - 1;
     seurvaih = vaihe + 1;
     }
...
// Seuraava if-else -rakenne määrittää mihin vaiheeseen siirrytään
// sekä sen kuinka monetta kertaa toistetaan do-while -lausetta.
if (toiminto == 0) alkuun();
else if (toiminto == 1) {
        edvaihe(edvaih);
        if (vaihe == 7) toisto[0]--;
        }
else if (toiminto == 2) {
        if (vaihe == 4) {
           if (document.getElementById('max').value == "") {
              alert("Ohjelma pysähtyi odottamaan syötettä, eikä
              etene ennen kuin olet syöttänyt luvun.");
              }
           else {
              var maks =
              parseInt(document.getElementById('max').value, 10);
              document.getElementById('max').value =
              maks.toString();
              seurvaihe(seurvaih);
              }
           }
        else {
           seurvaihe(seurvaih);
           if (vaihe == 8) toisto[0]++;
           }
        }
```

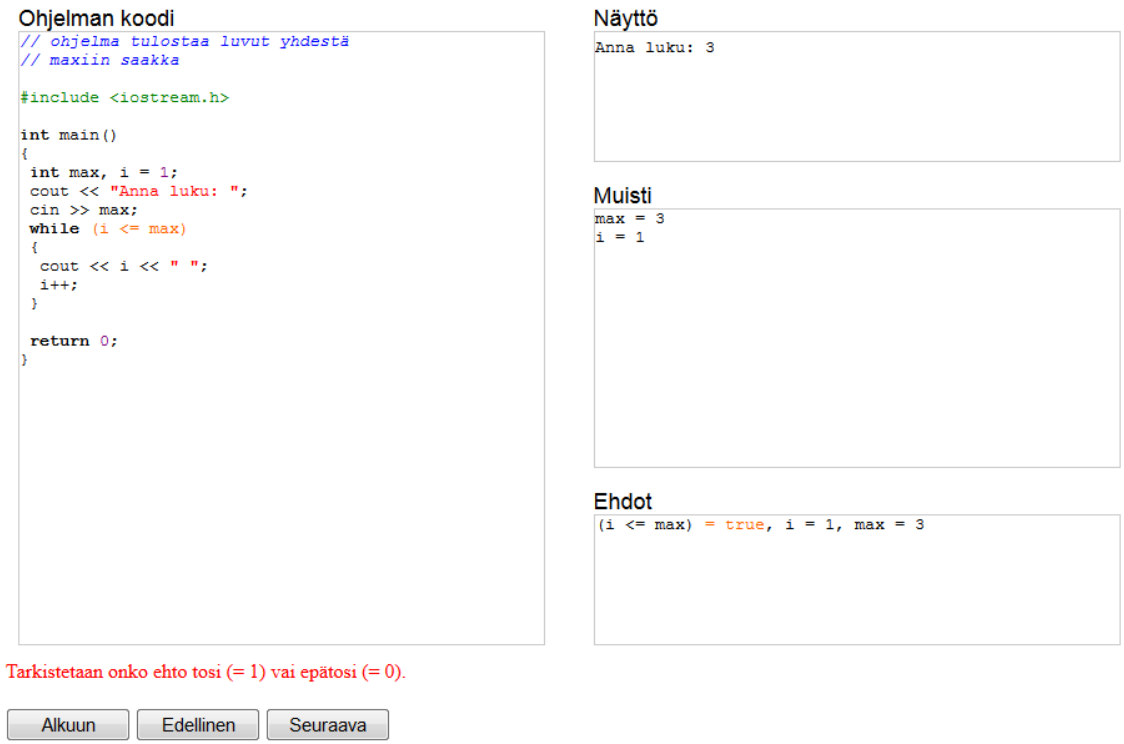
for-rakenteessa samassa lauseessa ehdon kanssa on myös alustettu muuttuja sekä toiminto, jonka myötä muuttujan arvo vähenee tai kasvaa jokaisen läpikäynnin jälkeen. Kun taas muissa toistorakenteissa ehtojen käyttämät muuttujat ja niiden arvojen muuttuminen ovat ohjelmoitu erikseen.

Kuvassa 11 on for-rakenteella toteutettu versio kuvan 10 simulaatiosta. Muuttujan i luominen ja alustaminen sijoittuvatkin ehdon tarkistamisen ohella sulkeiden sisään. Jos ehto on epätosi, käydään läpi for-silmukan sisältö. Sen jälkeen kasvatetaan muuttujan i arvoa ja käydään silmukka muuttujan alustamista lukuun ottamatta uudestaan läpi.



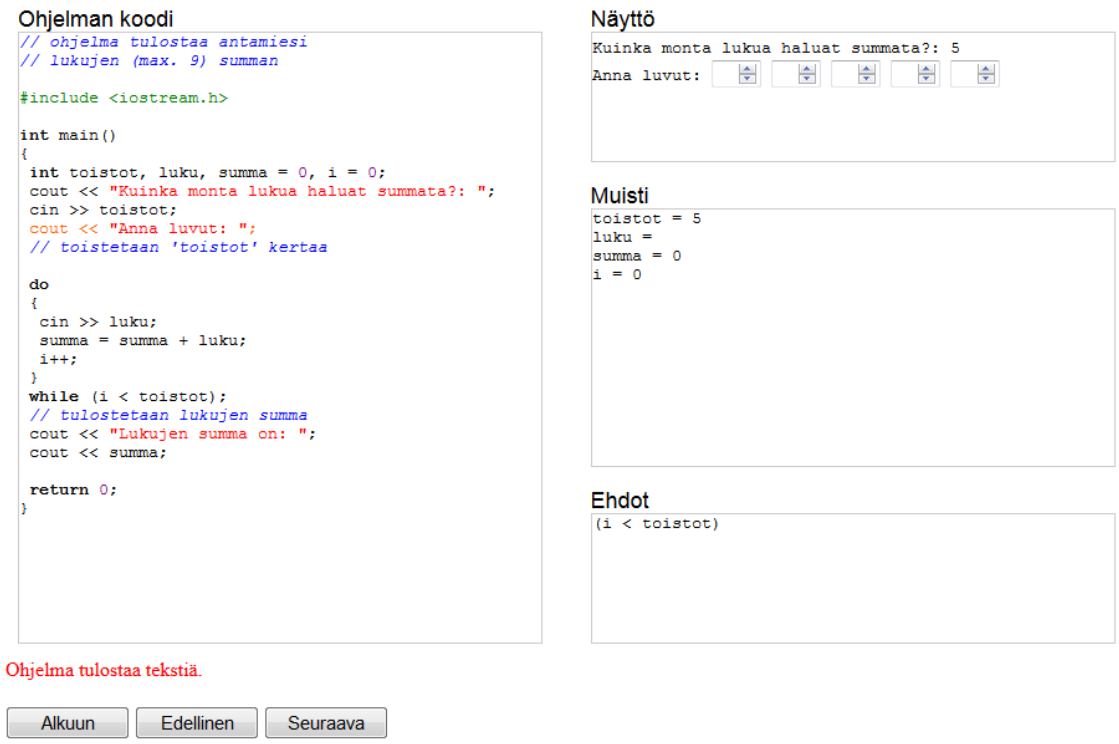
KUVA 11. Ehdon käsittely for-rakenteessa

while-rakenteen ero do-while -rakenteeseen on se, että while-rakenteessa ehto käsitellään jo silmukan alussa ja aina silmukan päätyttyä palataan sinne. Kuvassa 12 näkymä simulaatio toimii toistorakenteen muotoa lukuun ottamatta aivan kuin ensimmäisen kuvan simulaatio.



KUVA 12. Ehdon käsittely while-rakenteessa

Osassa simulaatioista joidenkin muuttujien arvot syötetään toistolauseiden sisällä. Kuten kuvassa 13 näkyy, C++-simulaatioissa syötekentät tulevat näkyviin jo toistolausetta edeltävässä kohdassa, mutta kukin arvo tallennetaan muistiin vasta cin-operaattoria käsittelevässä vaiheessa. Kuten aiemmin todettu, yhden cin-operaattorin käytönkin aikana on mahdollista syöttää arvot useammalle muuttujalle. Mutta Javassa voi syöttää vain yhden muuttujan arvon kerrallaan. Ja toistolauseessa annetaan muuttujalle luku uusi arvo jokaisella toistokerralla.



KUVA 13. Syötteen antaminen C++-simulaatiossa toistorakenteen kautta

4.1.4 Taulukot

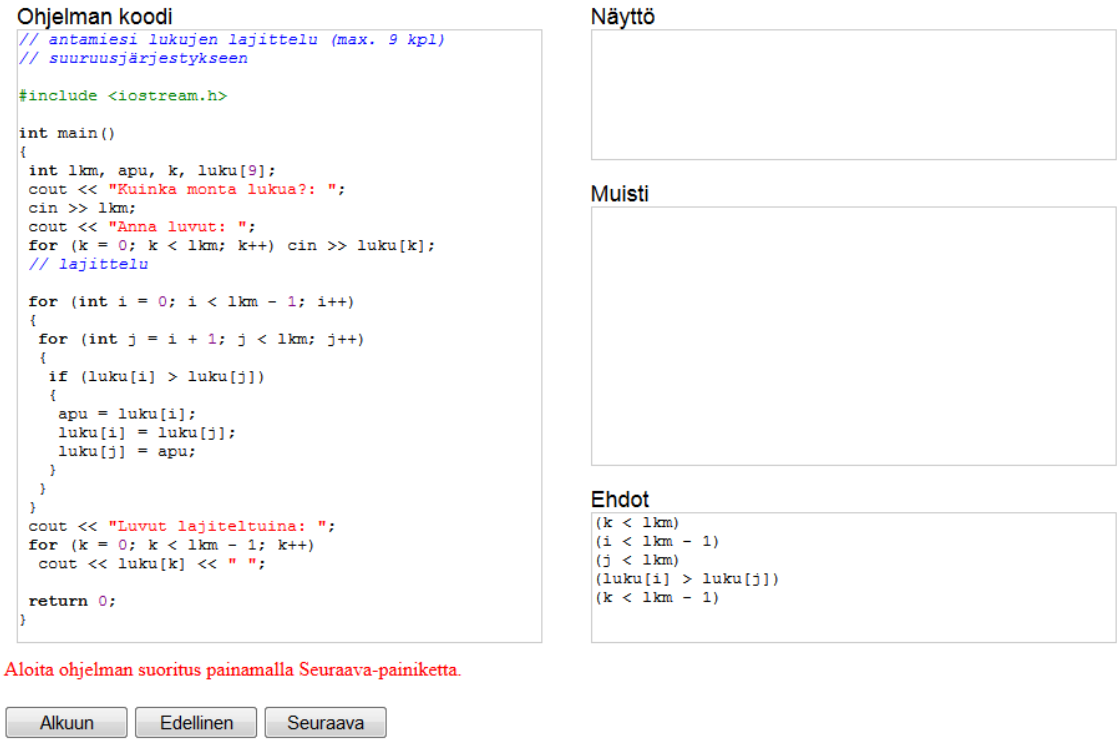
Taulukoiden käsittelyssä erikoisinta on se, miten ne merkitään muistiin. Kuten kuvista 14 ja 15, taulukon alkiot ovat erotettu toisistaan pilkuilla. C++-materiaali sisältää myös eimerkkejä char-taulukoiden käytöstä. Esimerkki char-taulukon merkinnästä muistiin on nähtävissä kuvasta 15.

KUVA 14. 10-alkioisen int-taulukon alustaminen Java-simulaatiossa

KUVA 15. 15-alkioisen char-taulukon alustaminen C++-simulaatiossa

Erikoistapauksena muista taulukoita käsittelevästä simulaatiosta se, mikä näkyy kuvassa 16. Siinä käyttäjän on ensin syötettävä enintään 9 kokonaislukua taulukon alkioiksi.

Sitten taulukon alkioiden järjestystä muutetaan sisäkkäisiä for-rakenteita käyttäen, ja lopuksi tulostetaan taulukon alkiot lajittelun jälkeen.



KUVA 16. Taulukon alkioita lajitteleva simulaatio

Taulukon alkioiden järjestyksen muutoksen näyttämiseksi simulaation JavaScript-osioon on sisällytettynä muuttujat, jotka ovat lueteltuna esimerkissä 10. Ehdot näiden muuttujien arvojen muuttumiselle esimerkissä 11.

ESIMERKKI 11. Kuvan 16 simulaation lähdekoodin käyttämien muuttujien alustus

```
var taulu = new Array();
var alkioI = new Array();
var alkioJ = new Array();
var apu = new Array();
var apulaskuri = -1;
```

ESIMERKKI 12. Kuvan 16 simulaation vastine esimerkin 4 tapaukselle.

```
if (toiminto == 0) alkuun();
else if (toiminto == 1) {
        if (vaihe == 20) {
            alkioI[apulaskuri] = 0;
            alkioJ[apulaskuri] = 0;
            apu[apulaskuri] = 0;
            apulaskuri--;
            }
        else if (vaihe == 22) {
                taulu[toisto[2]] = taulu[toisto[1]];
                taulu[toisto[1]] = apu[apulaskuri];
                }
        else if (vaihe == 23) toisto[2]--;
        else if (vaihe == 24) toisto[1]--;
        else if (vaihe == 25) {
                var lkm =
                parseInt(document.getElementById('lkm').value);
                toisto[0] = lkm;
                }
        edvaihe(edvaih);
        if (vaihe == 10 || vaihe == 29) toisto[0]--;
        }
else if (toiminto == 2) {
        ...
        seurvaihe(seurvaih);
        if (vaihe == 11 || vaihe == 30) toisto[0]++;
        else if (vaihe == 20) {
                apulaskuri++;
                alkioI[apulaskuri] = toisto[1];
                alkioJ[apulaskuri] = toisto[2];
                apu[apulaskuri] = taulu[toisto[1]];
                }
        else if (vaihe == 22) {
                taulu[toisto[1]] = taulu[toisto[2]];
                taulu[toisto[2]] = apu[apulaskuri];
                }
        else if (vaihe == 23) toisto[2]++;
        else if (vaihe == 24) toisto[1]++;
        else if (vaihe == 25) toisto[0] = 0;
        }
```

Taulukon toisto alkion 0 arvo on yhtä kuin kuvan 31 simulaatiossa muuttujan k arvo ja alkioiden 1 ja 2 arvot yhtä kuin muuttujien i ja j arvot. Taulukkoon taulu tulee sisällytetyksi simulaatioon syötetyt taulukon luku alkiot. Taulukoiden alkioI ja alkioJ alkiot ovat samat kuin arvot muuttujilla i ja j siinä tapauksessa, kun for-rakenteiden sisällä olevan if-lauseen ehto on tosi. Silloin muuttujan apulaskuri arvoa kasvatetaan yhdellä ja sen muuttujan arvon indeksiin taulukkossa apu tallennetaan mikä if-lauseen sisällä olisi kuvan 31 simulaation muuttujan apu arvo.

4.1.5 Funktiot ja aliohjelmat

Merkittävin poikkeus funktioissa ja aliohjelmissa pääohjelmaan verrattuna on, että funktioista poistuessa sen parametrien ja sen sisällä luodut muuttujat pysyvät muistissa ainoastaan niin kauan kuin kyseistä funktiota käsitellään.

KUVA 17. Aliohjelman kutsuminen.

KUVA 18. Aliohjelma palauttaa arvon muuttujalle tilavuus.

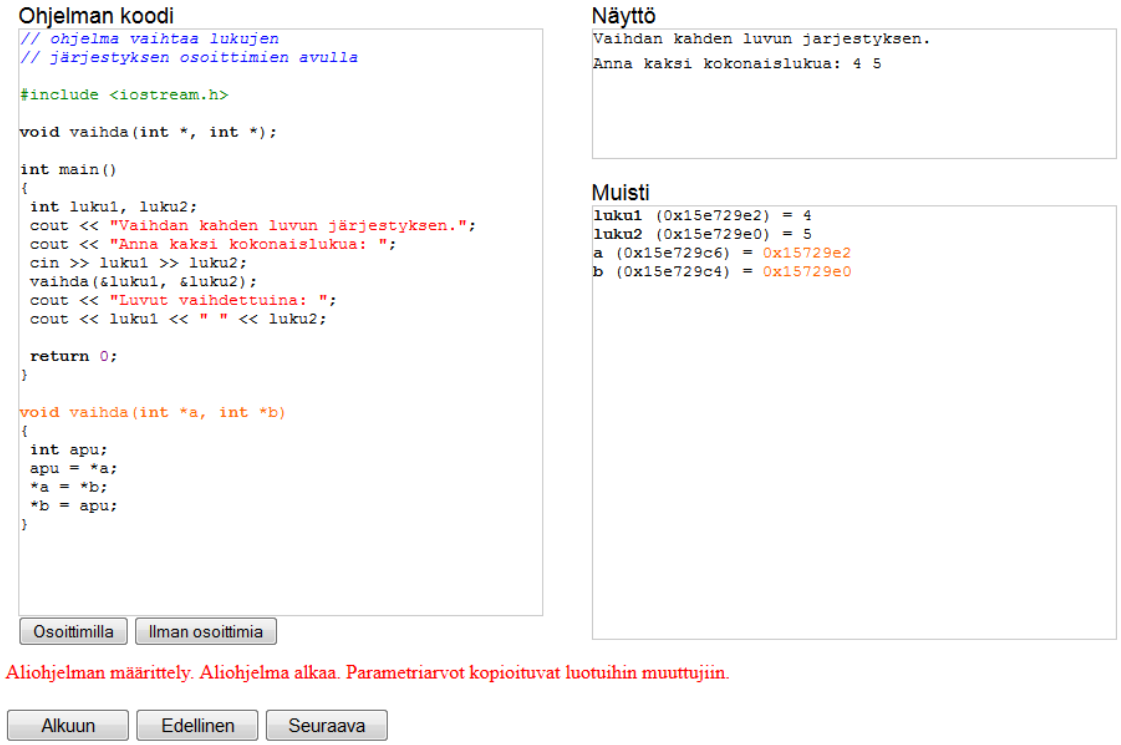
Kuvan 17 tilanteessa kutsutaan aliohjelmaa kuutio ja sen parametrina välitetään muuttujan sivu arvo. Muuttuja x on tallennettuna muistiin ainoastaan niin kauan kuin aliohjelman käsittely on kesken. Kuvassa 18 aliohjelman ajo on päättynyt ja muuttujalle tilavuus on palautettu arvona muuttujan sivu kuutio. Muuttuja x on hävitetty muistista.

4.1.6 Osoittimet ja viittaukset

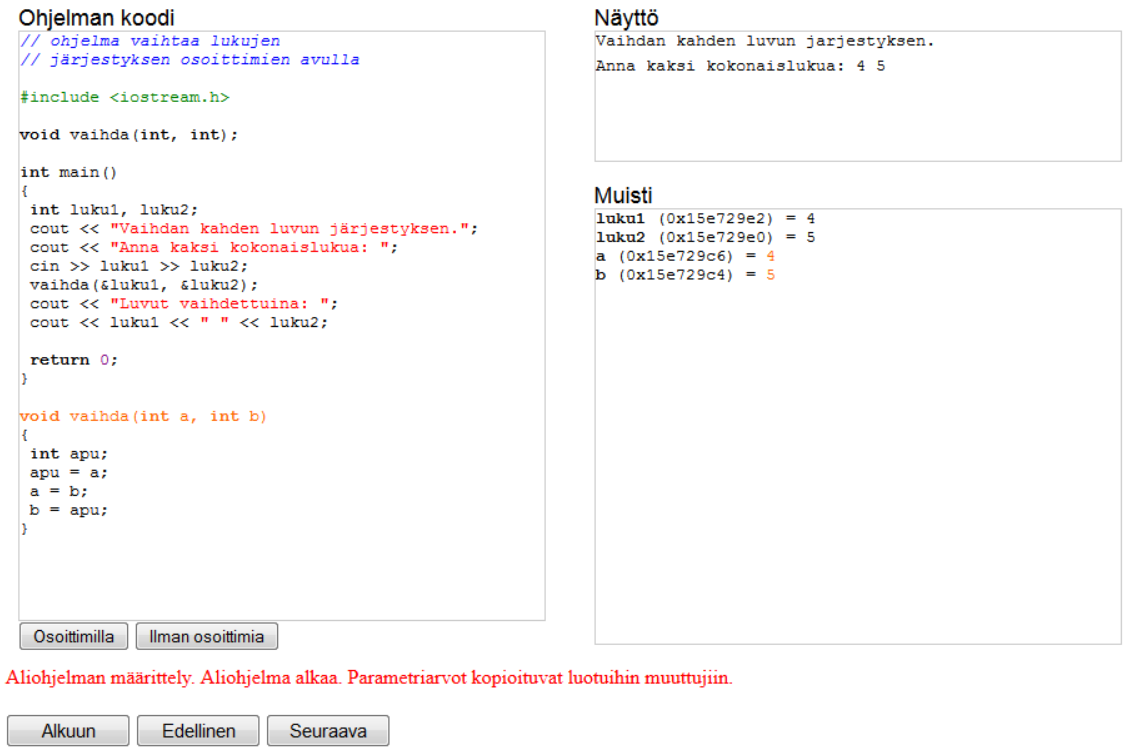
Osoittimia ja viittauksia käyttävissä simulaatioissa on muisti-osiossa näytettynä muuttujien lisäksi myös niiden osoitteet. Nämä simulaatiot ovat sisällytettynä vain C++-materiaaliin.

Kuvissa 19 ja 20 nähtävä simulaatio on siitä erikoinen, että siinä käyttäjä voi valita käytetäänkö osoittimia vai ei. Kuvan 19 tilanteessa osoittimet ovat käytössä ja aliohjelmassa vaihda() muuttujat a ja b saavat arvoikseen muuttujien luku1 ja luku2 osoitteet. Kuvan 20 tilanteessa ne saavatkin arvoikseen muuttujien luku1 ja luku2 arvot. Mutta jos simulaatiossa on tarkoitus vaihtaa näiden kahden muuttujan arvot keskenään, käyttäjän tulee käyttää osoittimia. C++-kielessä, jos lukumuuttujan arvo muutetaan aliohjelmassa,

muutos säilyy pääohjelmaan palaamisen jälkeen vain, jos ne ovat välitetty osoittimien avulla. Kuvassa 19, kun muuttujien arvot tulostetaan uudelleen, ne ovat vaihtuneet myös pääohjelmassa. Mutta kuvassa 20 vaihtuminen jää aliohjelman sisäiseksi tapahtumaksi.



KUVA 19. Osoittimien välitys aliohjelmaan

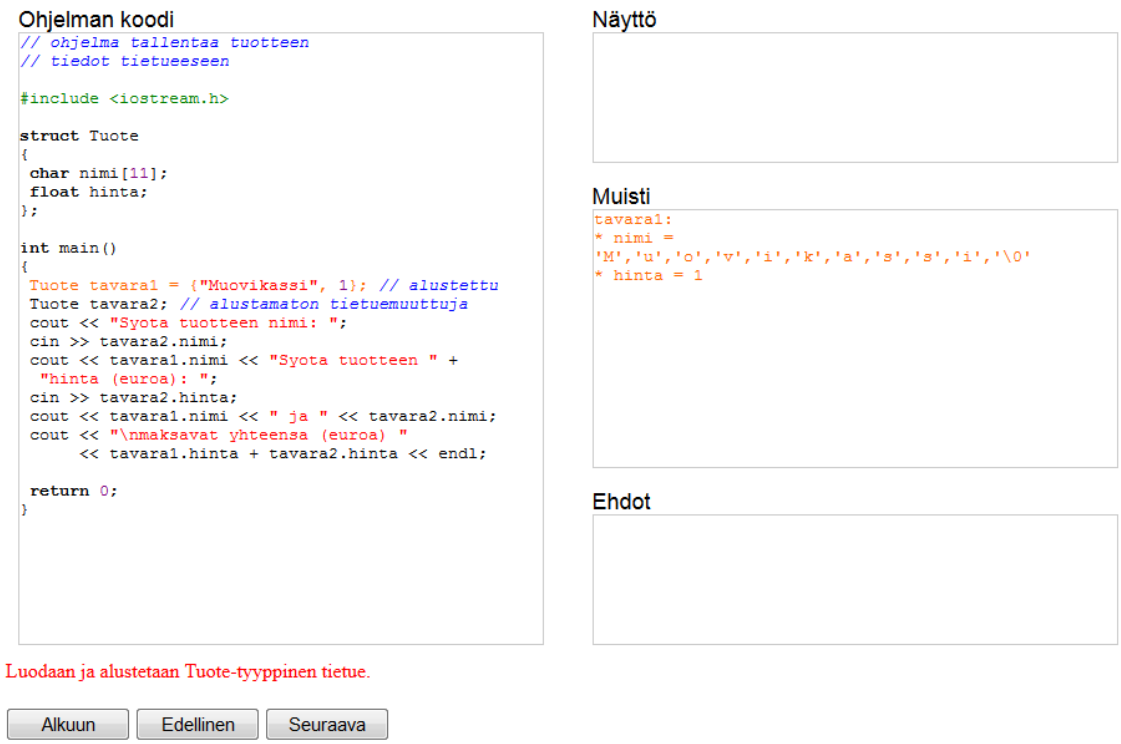


KUVA 20. Kuvan 19 simulaatio ilman osoittimia

4.1.7 Tietueet

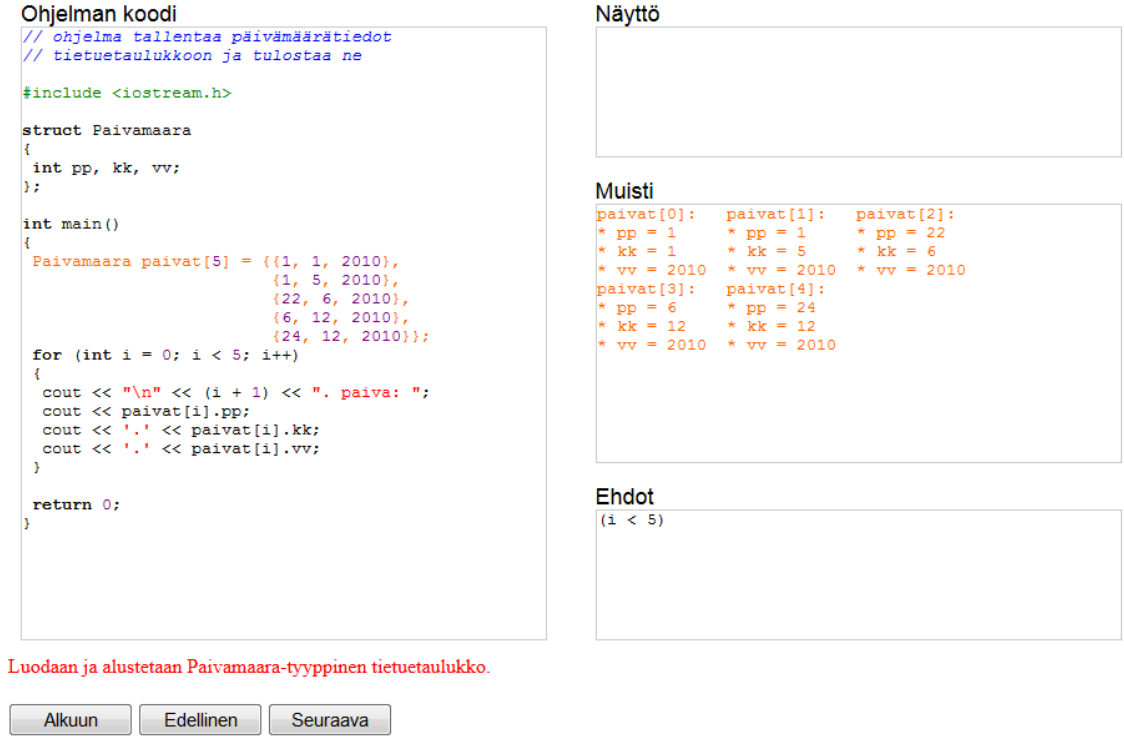
Tietueita käytetään siinä tapauksessa, jos johonkin asiaan liittyy monta eri arvoa. Toisinaan nämä arvot ovat jopa keskenään erityyppisiä muuttujia. Tietueen tietotyypin nimi sekä tietueeseen kuuluvien muuttujien nimet määritellään aina ennen pääohjelmaa.

Kuvan 21 tilanteessa on tietue (*struct*), jossa on tietuemuuttujina char-taulukko, jossa voi olla enintään 11 alkiota, sekä float-muuttuja eli desimaaliluku. Kuvassa Tuote-tyyppiselle tietueelle tavara1 määritellään tietuemuuttujien nimi ja hinta arvoiksi merkkijono ”Muovikassi” sekä lukuarvo 1. Simulaation muistikentässä tietue tavara1 näytetään siten, että tietueen nimen alle on lueteltuna sen muuttujien arvot.



KUVA 21. Luodaan Tuote-tyyppinen tietue, joka on jo valmiiksi alustettu.

Kuten kuvasta 22 näkyy, myös tietueista voi koota taulukoita, mutta tietuetaulukossa yksi alkio sisältääkin useamman muuttujan arvot. Siksi tietuetaulukko on simulaatiossa merkitty muistikentän sisään vain luettelemalla kukin tietuetaulukon alkio erikseen.

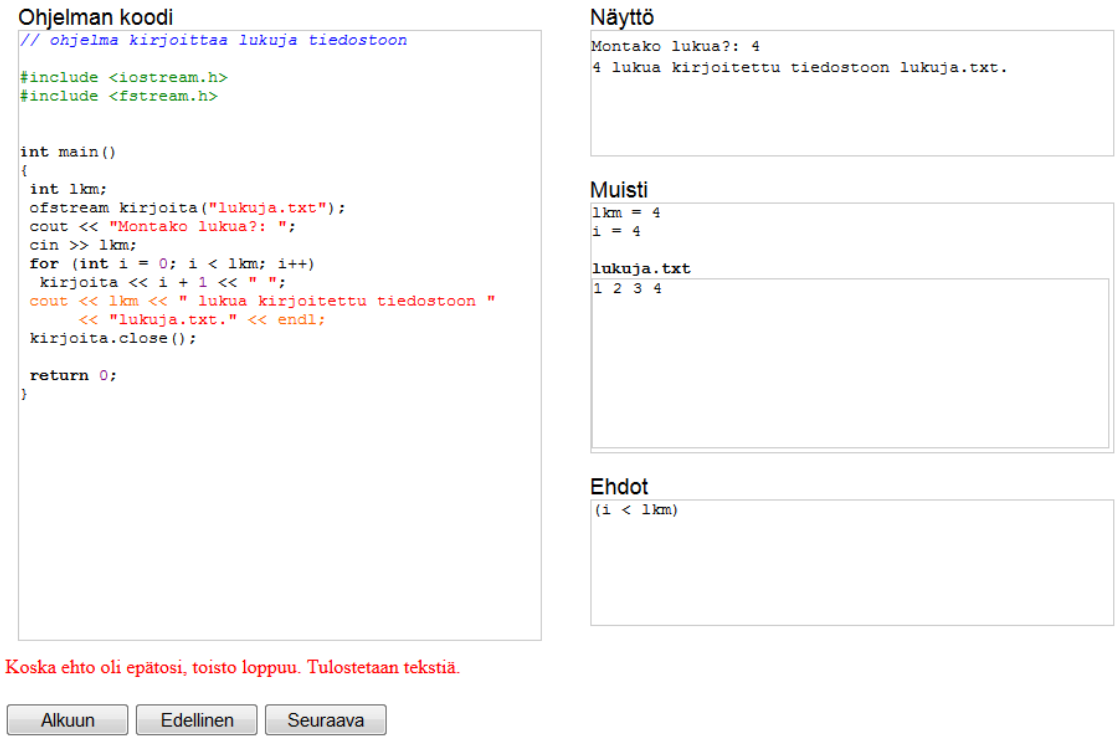


KUVA 22. Luodaan viiden alkion tietuetaulukko

4.1.8 Tiedostonkäsittely

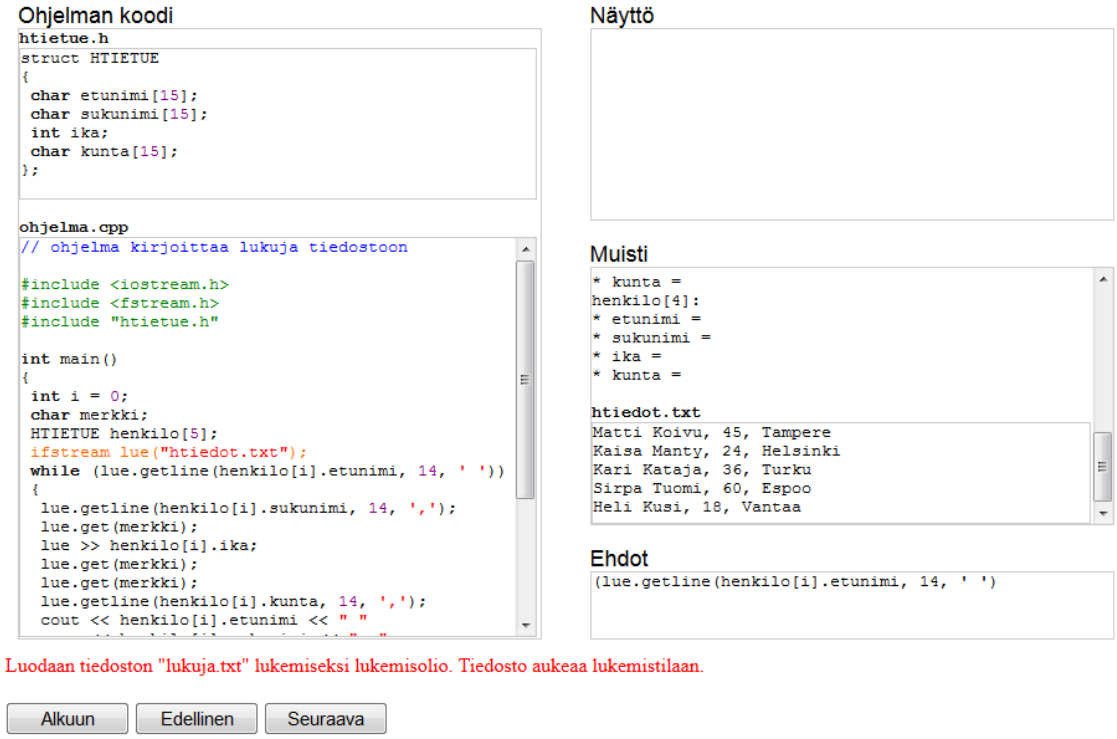
Jotkut ohjelmat voivat muuttaa toisten tiedostojen sisältöä. C++-materiaalissa on kaksi simulaatiota, joista toinen kirjoittaa sisältöä tekstitiedostoon ja toinen lukee sisältöä tekstitiedostosta. Näissä simulaatioissa tekstitiedostojen sisältö on käyttäjälle nähtävissä simulaation muistin kautta.

Kuvassa 23 näytetyn simulaatiossa kirjoitetaan lukuja alkaen luvusta 1, kunnes muuttujan i arvo on yhtä kuin muuttujan lkm arvo. Näkymä tekstitiedostosta lukuja.txt on tullut näkyviin muistinäkymän kohdalle näkyviin ofstream-oliota luotaessa. Mutta sen on tarkoitus poistua näytöltä, kun tiedosto suljetaan close()-funktion kohdalla. Luvut 1–4 on kirjoitettu tiedostoon for-lauseen kautta.



KUVA 23. Tilanne, jossa luvut 1–4 on kirjoitettu tekstitiedostoon

Kuvan 24 tilanteessa avataan tekstitiedosto htiedot.txt. Sen sisältö on tarkoitus tallettaa HTIETUE-tyyppisistä tietueista koostuvaan taulukkoon while-lauseeseen sisällytettyjen getline-funktioiden kautta. Kyseinen simulaatio on C++-materiaalin ainut, joka perustuu kahdesta erillisestä tiedostosta koostuvaan ohjelmaan. Tietueen HTIETUE määrittely löytyy erillisestä otsikkotiedostosta htietue.h.

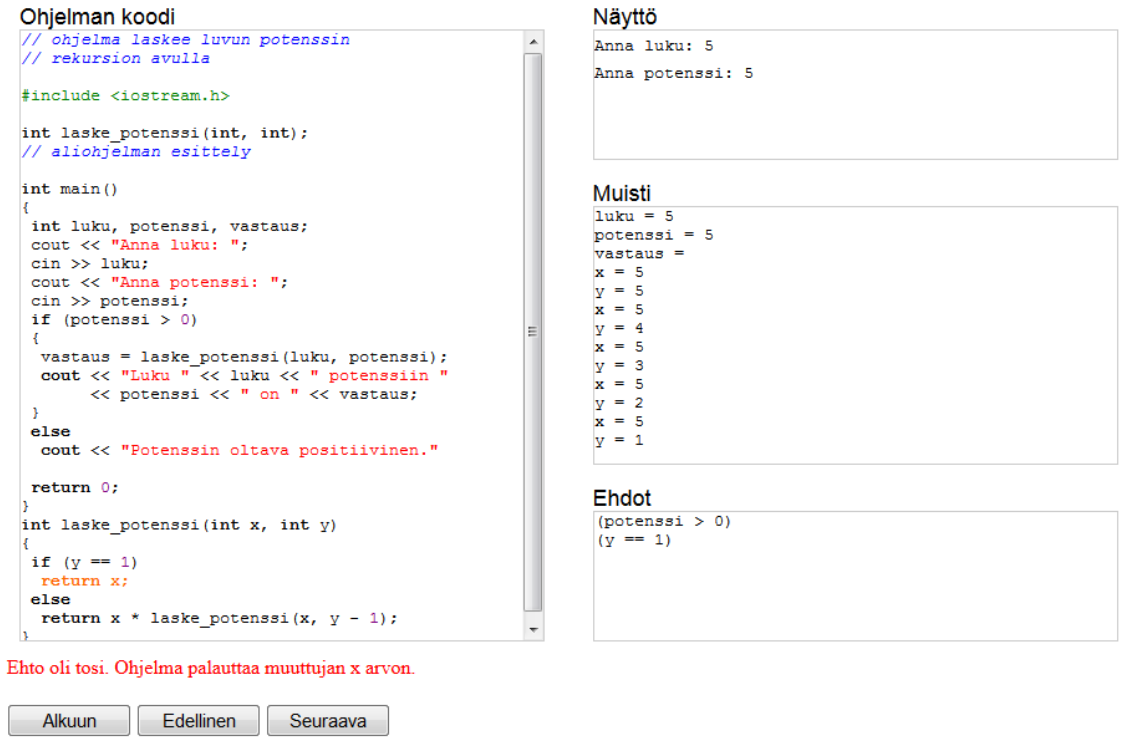


KUVA 24. Tekstitiedoston avaaminen sen lukemista varten

4.1.9 Rekursio

Materiaalissa oli yksi ainoa C++-simulaatio, joka havainnollisti käyttäjälle rekursiota. Rekursiossa aliohjelma kutsuu itse itseään uudestaan. Tästä on hyötyä esimerkiksi potenssilaskuissa tai yhtälöiden ja funktioiden laskennassa.

Kuvan 25 simulaatiossa on kutsuttu jo viidettä kertaa samaa aliohjelmaa. Ja aliohjelman laske_potenssi() parametrien x ja y arvot ovat lueteltuna muistikentässä viisi kertaa. Mutta muuttujan y arvo on jokaisella toistokerralla yhden arvon verran pienempi. Kun arvo y on 1, aliohjelman kutsuminen uudelleen päättyy. Kuvassa 25 ollaan nyt vaiheessa, jossa on tarkoitus palauttaa muuttujan x arvo aliohjelman edelliseen ajoon. Kuvassa 26 nähdään mitä muuttujan x arvon palautuksen jälkeen on tapahtunut. Siinä on palattu kohtaan, jossa aliohjelmaa kutsuttiin uudestaan ja nyt sen arvo kerrottuna muuttujan x arvolla palautetaan ohjelman edelliseen ajoon. Tämän seurauksena muuttujan x arvo kerrotaan yhä jatkuvasti palautetuilla arvoilla, kunnes palataan pääohjelmaan.



KUVA 25. Potenssilaskuja laskeva simulaatio

KUVA 26. On palattu aliohjelman edelliseen ajoon kohdan *return x* jälkeen.

Kuvassa 27 aliohjelman ensimmäinen ajo on päättynyt. Jokaisen *returnin* kautta palautettu arvo on kerrottu uudelleen parametrinä x välitetyllä muuttujan luku arvolla. Ensimmäisen ajon palauttama arvo on yhtä kuin 5 potenssiin 5 eli luku 5 kerrottuna 5 ker-

taa itsellään. Muuttujan vastaus arvoksi palautetaan koko laskutoimituksen tulos eli luku 3125.

KUVA 27. Aliohjelman toistojen jälkeen

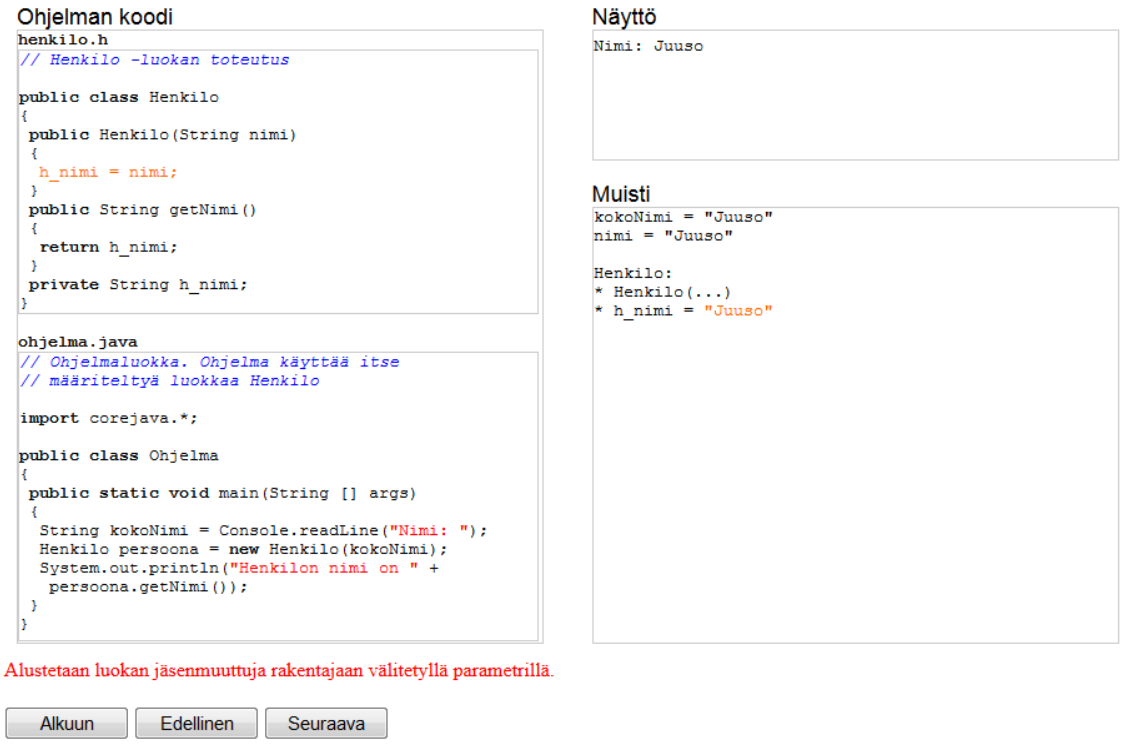
Liitteessä 3 on nähtävissä kuvien 25–27 simulaation HTML-dokumentista.

Kuten yllä olevassa koodissa näkyy, toisto-taulukon alkioita voidaan käyttää muuhunkin vain toistolauseiden toistokertojen laskentaan. Tässä tilanteessa alkion toisto[0] arvoa vähennetään yhdellä aina, kun aliohjelmaa kutsutaan, ja alkion toisto[1] arvoa vähennetään yhdellä aina, kun aliohjelmasta palautetaan arvo, niin kuin vaiheessa 15 (*return x*) tai vaiheessa 18 (*return x * laske_potenssi(x, y - 1)*).

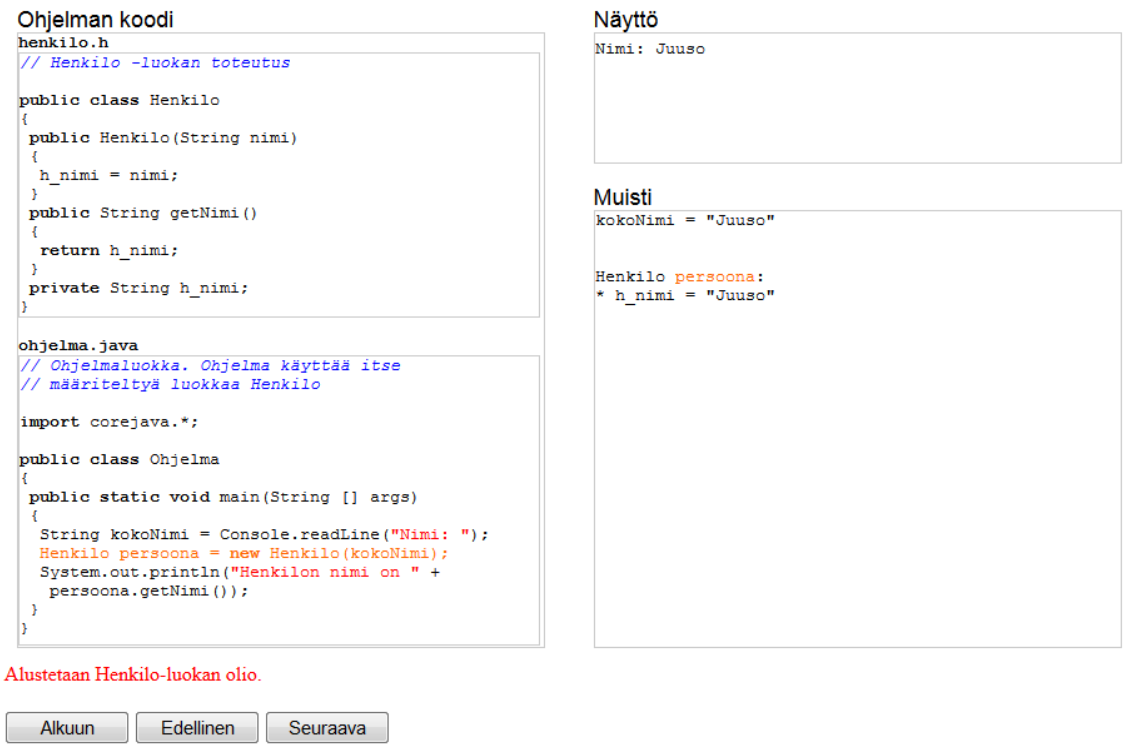
4.1.10 Oliot

Olio-ohjelmointia havainnollistavia simulaatioita on tässä työssä sisällytettynä ainoastaan Javan opetusmateriaalissa. Olioiden ero tietueista on yleensä siinä, että siihen voi sisällyttää muuttujien lisäksi myös funktioita. Tietyn oliojoukon piirteet määritellään luokan (*class*) kautta.

Kuvan 28 tilanteessa on syötetty merkkijono ja välitetty se parametrina Henkilo-luokan rakentajaan. Rakentajan tehtävä on edesauttaa alustamaan olion attribuutit parametreina välitettyjen muuttujien arvoilla. Kuvassa 29 rakentaja on tehnyt työnsä ja merkkijono ”Juuso” on välitetty Henkilo-luokan olion persoona attribuutin eli jäsenmuuttujan h_nimi arvoksi.

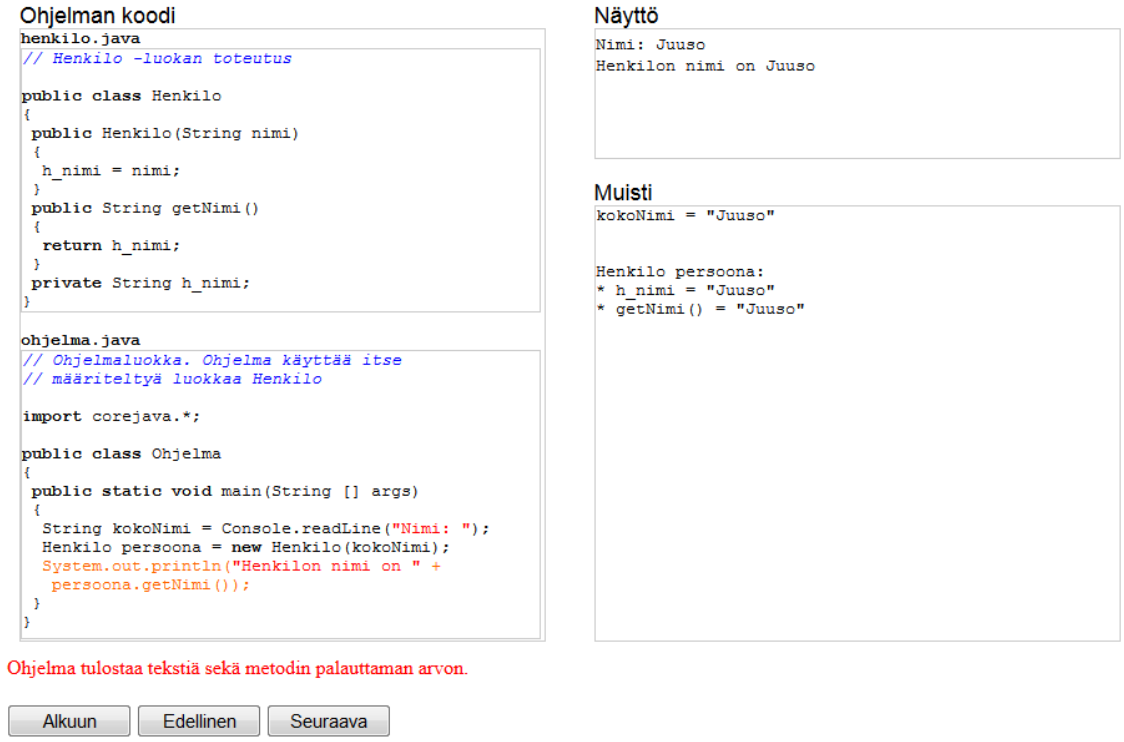


KUVA 28. Henkilo-luokan olion attribuutin h_nimi alustaminen



KUVA 29. Todetaan Henkilo-luokan olio alustetuksi.

Kuvassa 30 on tulostettu tekstiä sekä luokan persoona attribuutin h_nimi arvo. Koska attribuutti h_nimi on private eli yksityinen, siihen pääsee käsiksi vain luokan metodien kautta. Sen arvo saadaan pääohjelman käyttöön metodin getNimi() avulla sen antaessa palautusarvona attribuutin arvon.



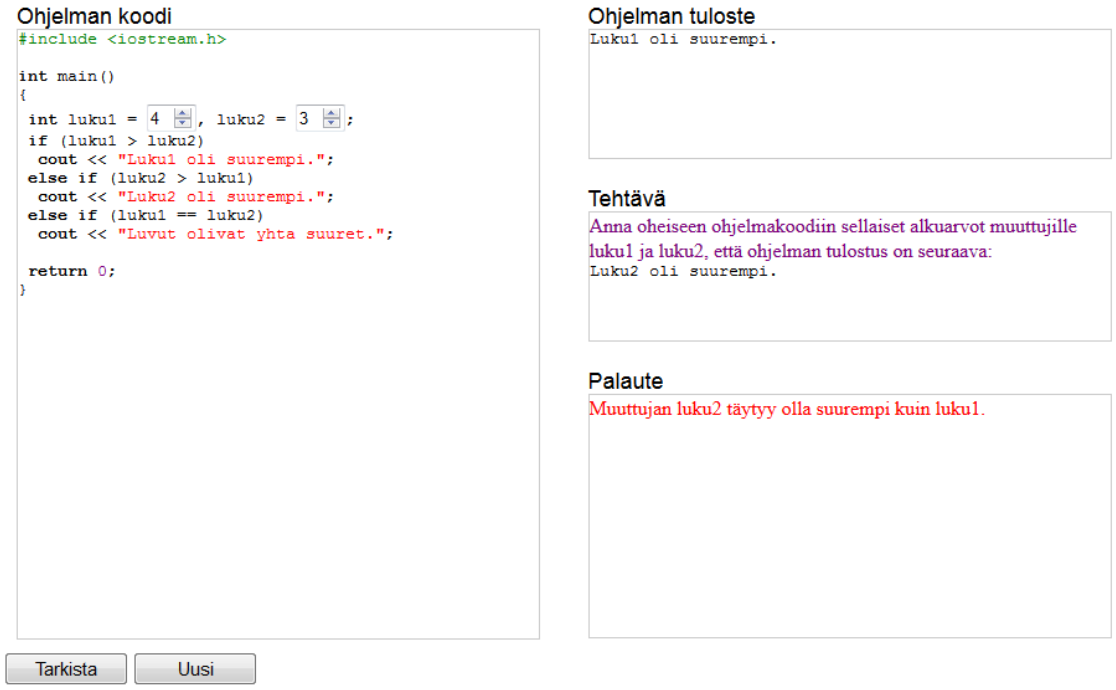
KUVA 30. Tulostetaan Henkilo-luokan olion attribuutti metodin avulla

4.2. Harjoitussimulaatiot

Harjoitussimulaatiot ovat sisällytettynä ainoastaan C++:n opetusmateriaaliin. Niissä käyttäjän on annettava arvot tyhjiin kohtiin käsiteltävässä ohjelmakoodissa. Arvojen täytyy olla sellaiset, että ne täyttävät tehtävänannossa määritellyt ehdot.

Harjoitussimulaatioissa on vain kaksi painiketta: *Tarkista* ja *Uusi*. Painikkeen *Tarkista* painaminen kutsuu funktiota ratkaisu(), joka tarkistaa käsiteltävän ohjelmakoodin (kirjoitettuna HTML-dokumentissa tekstin ”Ohjelman koodi” jälkeisen div-osion sisään) sekaan asetettujen syötekenttien arvot. Jos ne ovat tyhjiä tai eivät ole kentän hyväksymässä muodossa, annetaan käyttäjälle ilmoitus tästä JavaScriptin alert-funktiota käyttäen. Muussa tapauksessa ”Ohjelman tuloste”-näkymään tulostuu oikean C++-ohjelman toiminnan mukaisesti tekstiä ja ”Palaute”-näkymään ilmestyy tekstiä, joka ilmoittaa antoiko käyttäjä syötekenttiin oikeat arvot. Painike *Uusi* kutsuu funktiota arvonta(), joka tyhjentää syötekenttien arvot sekä ohjelman tulosteen ja kutsuu tiedostosta simulaatio.js funktiota, jonka arpoma numero muuttaa tehtävänantoa antaen käyttäjälle mahdollisuuden kokeilla samaa simulaatiota toisenlaisella tehtävänannolla.

Kuvassa 31 nähtävässä simulaatiossa on pyydetty arvoja muuttujille luku1 ja luku2. Käyttäjä on syöttänyt arvot 4 ja 3. Vastaus on väärin, koska ohjelma tulostikin ”Luku1 oli suurempi”, vaikka sen pitäisi tulostaa ”Luku2 oli suurempi”. Kuvan simulaatio on sama kuin kuvan 3 simulaatio, ja sen lähdekoodi on katsottavissa liitteestä 2.



KUVA 31. Palaute väärästä vastauksesta harjoitussimulaatiossa

Nykyinen tehtävänanto on kirjoitettu tiedostoon valmiiksi ja tehtävänannon tyyli määräytyy muuttujan tehtava[0] arvon mukaan. Kuvan 31 tapauksessa se on 2. Jos arvo olisi 1, tehtävänanto olisi, että muuttujan luku1 pitäisi olla suurempi kuin luku2. Jos arvo taas olisi 3, molempien muuttujien pitäisi olla yhtä suuret. Painikkeen *Uusi* painaminen kutsuu HTML-dokumentissa olevaa funktiota arvonta(), jossa kutsutaan tiedostossa simulaatio.js olevaa funktiota uusiTeht(). Funktion koodi on nähtävissä esimerkistä 13.

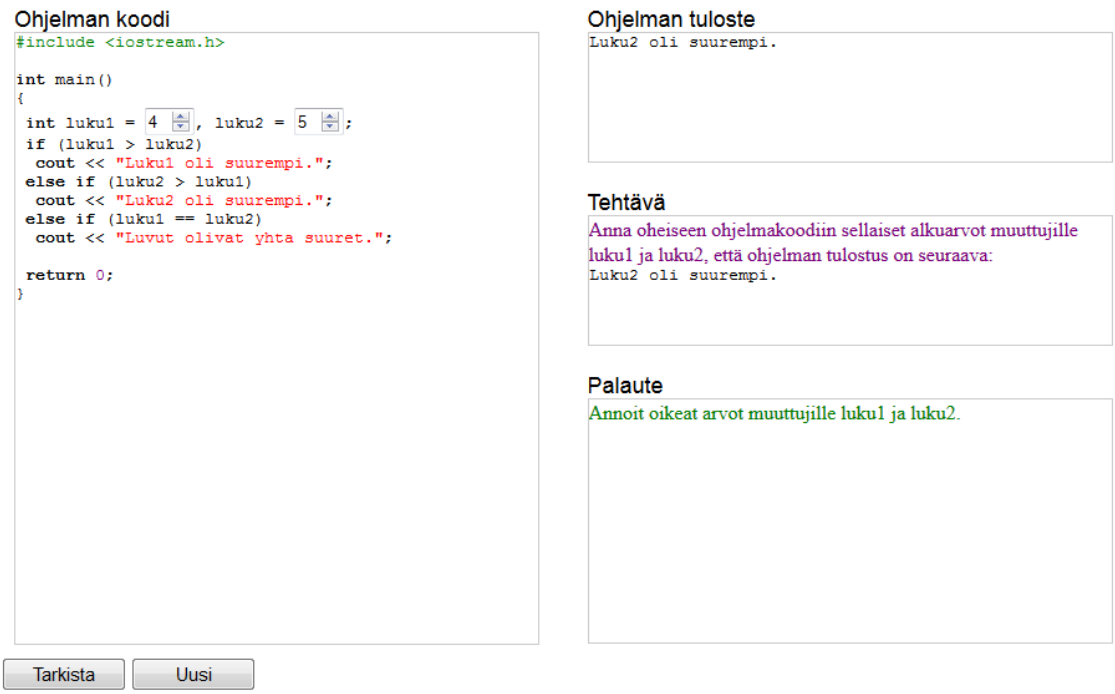
ESIMERKKI 13. Funktio uusiTeht() tiedostosta simulaatio.js

```
function uusiTeht(min, max) {
    tarkistus = false;
    document.getElementById("palaute").innerHTML = "";
    // Korvataan uusi tehtävä vanhalla arpomalla numero,
    // joka palauttaessaan arvon HTML-tiedostoon
    // määrittää tehtävän ehdon.

    return Math.floor(Math.random() * (max - min + 1) + min);
}
```

Kuvan 31 simulaation tapauksessa parametrin min arvon on oltava 1 ja parametrin max arvo 3. Funktio tyhjentää simulaation palauteen ja palauttaa näiden kahden arvon väliltä arvotun luvun.

Kuvassa 32 käyttäjä on antanut muuttujalle luku1 arvon 4 ja muuttujalle luku2 arvon 5 sekä napsauttanut hiirellä painiketta *Tarkista*. Muuttujan luku2 arvo on suurempi, joten tehtävä on oikein. Kun harjoituksen antama tehtävä tarkistetaan, kutsutaan HTML-dokumentin funktiota ratkaisu(). Tarkistetaan onko käyttäjä syöttänyt arvoja ollenkaan tai ovatko syötteet kelvollisia. Palautteen tekstin väri ja muoto määräytyvät tiedoston simulaatio.js funktion tarkista() kautta.



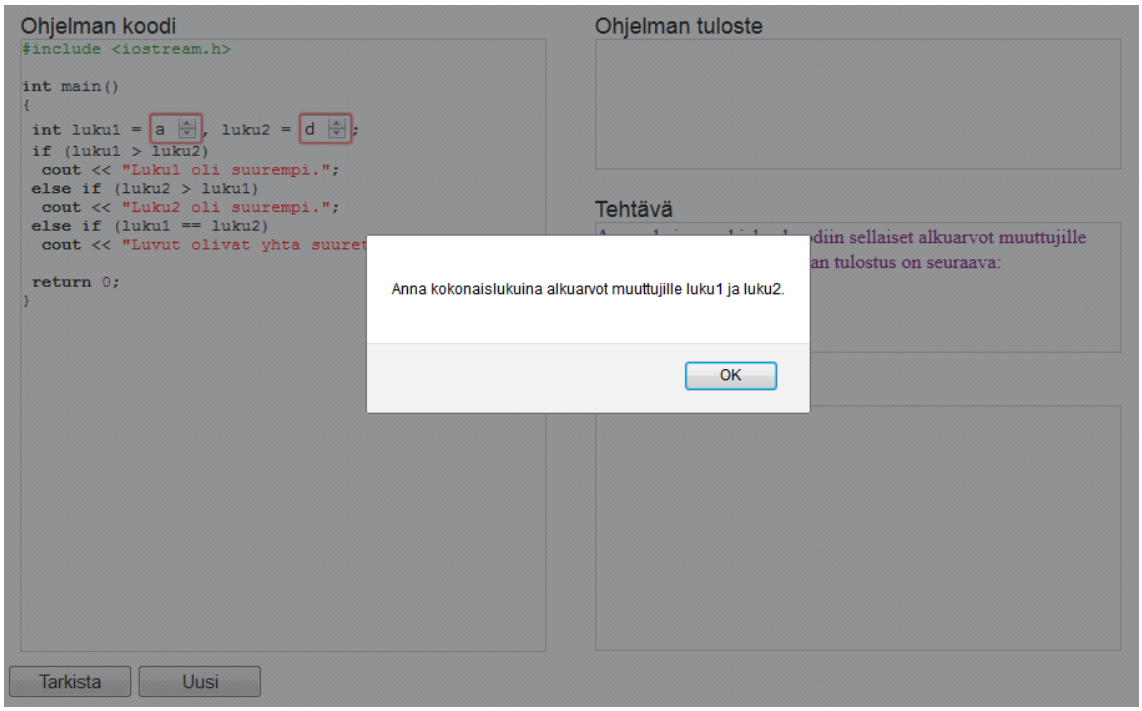
KUVA 32. Palaute oikeasta vastauksesta harjoitussimulaatiossa

ESIMERKKI 14. Funktion tarkista()

```
function tarkista(arvo, vastaus) {
    tarkistus = arvo;
    if (arvo == true) {
       document.getElementById("palaute").innerHTML =
       document.getElementById("palaute").innerHTML +
       "<span style='color:green;'>" + vastaus + "</span>";
       }
    else
       document.getElementById("palaute").innerHTML =
       document.getElementById("palaute").innerHTML +
       "<span style='color:red;'>" + vastaus + "</span>";
       // Merkitsee HTML-dokumentin palaute-osioon
       // ilmoituksen siitä miten tehtävä onnistui.
}
```

Jos vastaus on oikea, muuttuja arvo on *true*. Tässä tapauksessa palauteteksti, joka on välitetty funktioon parametrin vastaus arvoksi, tulostetaan vihreän värisenä. Jos muuttuja arvon onkin *false*, palauteteksti tulostetaan punaisena.

Kuvassa 33 simulaatiossa molempien muuttujien on oltava kokonaislukuarvoja. Mutta käyttäjä syöttänyt molempiin kirjaimet, joten simulaatio ilmoittaa JavaScriptin alert-funktiota käyttäen, ettei syöte ole kelvollinen.



KUVA 33. Virheelliset syötteet harjoitussimulaatiossa

5 YHTEENVETO

Opetusmateriaali on tehokas opettamaan erityisesti niitä, jotka ovat innostuneempia oppimaan enemmän käytännön kuin teorian kautta. Harjoitusten kautta he näkevät täsmälleen mitä kunkin simulaation ohjelmakoodi tekee, jos se olisi toteutettu oikeana sovelluksena. Materiaalin käyttö on kuitenkin suositeltavaa kaikille, jotka näihin ohjelmointikieliin ovat tutustumassa.

Tiedostoihin simulaatio.js ja sheets.css sisältyvä osuus kokonaisuudesta jäi pienemmäksi kuin simulaatioiden omat toiminnallisuudet, mutta oli rakenteen kannalta järkevin ratkaisu.

Monien simulaatioiden lopulliset ratkaisut osoittautuivat työn aikana huomattavasti yksinkertaisemmiksi kuin mitä työtä aloitettaessa oli arvioitu. Paras esimerkki tästä on `<span>`- ja `<div>`-elementtien sisällön käsittely. Niiden alkuperäinen muoto vain kirjoitettiin HTML-dokumentin koodiin ja painikkeiden kautta kutsuttujen JavaScript-funktioiden kautta sisältö muuttui.

Päivitetty versio materiaalista vaatii tietokoneelta ja selaimelta vähemmän kuin alkuperäinen, sillä se käyttää vain standardin mukaista HTML5 -rakennetta ilman selaimeen tarvittavia laajennuksia. Tällöin kokonaisuus toimii lähes varmasti kaikissa tietokoneissa.

Luvussa 4.1.4 ja kuvassa 16 nähtävän simulaation toiminnallisuuden toteutus oli koko työn monimutkaisin osuus. Yksinkertaisinta oli toteuttaa kaikki sellaiset simulaatiot, joissa käyttäjän ei tarvinnut syöttää arvoja lainkaan. Mutta osassa niistäkin oli valinta- tai toistorakenteita, jotka olivat työläämpiä toteuttaa.

LÄHTEET

Adobe Director. Luettu 12.6.2014. http://www.adobe.com/products/director.html

W3C HTML Working Group. Luettu 9.4.2014. http://www.w3.org/html/wg/

Plan 2014. Luettu 11.5.2014.
http://dev.w3.org/html5/decision-policy/html5-2014-plan.html

JavaScript Overview - JavaScript | MDN. Luettu 12.5.2014.
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/JavaScript_Overview

New in JavaScript 1.8.5 - JavaScript | MDN. Luettu 30.6.2014.
https://developer.mozilla.org/en-US/docs/Web/JavaScript/New_in_JavaScript/1.8.5

Cascading Style Sheets. Luettu 30.6.2014. http://www.w3.org/Style/CSS/

Notepad++ Home. Luettu 12.5.2014. http://notepad-plus-plus.org/

LIITTEET

Liite 1. Kuvan 2 esimerkkisimulaation HTML-koodi

```
<html>
<script src="../simulaatio.js" type="text/javascript"></script>
<!-- Mahdollistaa simulaatio.js -tiedoston JavaScript-funktioiden
alkuun(), edvaihe(), seurvaihe() ja merkinta() käytön. -->
<head>
<meta charset="utf-8">
<title>Esimerkki</title>
<link rel="stylesheet" type="text/css" href="../sheets.css">
<script>
max = 13; // tämä muuttuja määrittää esimerkkikoodissa käsiteltävien
vaiheiden lukumäärän. Vaihtelee eri simulaatioissa.
function paivita(toiminto) {
    edvaih = vaihe - 1;
    seurvaih = vaihe + 1;
    if (vaihe != 0) {
       // Seuraava if-else -rakenne varmistaa, että oranssia
       // värimerkintää vailla oleva versio tekstistä säilyy,
       // vaikka ohjelma pysähtyisi syötteiden kanssa
       // ilmenemien ongelmien vuoksi.
       var luku = vaihe.toString();
       document.getElementById(luku).innerHTML = alkuperSisalto;
       }
    if (toiminto == 0) alkuun();
    else if (toiminto == 1) edvaihe(edvaih);
    else if (toiminto == 2) {
         if (vaihe == 5) {
           if (document.getElementById('luku1').value == "" ||
              document.getElementById('luku2').value == "") {
              alert("Ohjelma pysähtyi odottamaan syötettä, eikä etene
              ennen kuin olet syöttänyt kaksi kokonaislukua.");
              }
           else {
               var luku1 = parseInt
                      (document.getElementById("luku1").value, 10);
               var luku2 = parseInt
                      (document.getElementById("luku2").value, 10);
               // Jos syötteet eivät ole kokonaislukuja,
               // ohjelma poistaa niistä desimaalipilkun
               // jälkeen tulevat numerot.
               document.getElementById('luku1').value =
               luku1.toString();
               document.getElementById('luku2').value =
               luku2.toString();
               seurvaihe(seurvaih);
               }
           }
         else if (vaihe == 7) {
                 if (document.getElementById('desi1').value == "" ||
                 document.getElementById('desi2').value == "") {
                    alert("Ohjelma pysähtyi odottamaan syötettä, eikä
                    etene ennen kuin olet syöttänyt kaksi lukua.");
                    }
                 else seurvaihe(seurvaih);
                 }
```

```
      else seurvaihe(seurvaih);
      }
// Tarkistetaan funktion kautta voidaanko
//oranssi värimerkintä siirtää.
merkinta();

// Seuraava if-else -rakenne määrittää
// näkymän ohjelman tulosteesta.
if (vaihe > 4) {
   document.getElementById('tuloste1').innerHTML = "Anna kaksi
   kokonaislukua: ";
   if (vaihe == 5) {
    document.getElementById('luku1').style.visibility = 'visible';
    document.getElementById('luku2').style.visibility = 'visible';
    }
   else {
       document.getElementById('tuloste1').innerHTML =
       document.getElementById('tuloste1').innerHTML + " " +
       document.getElementById('luku1').value;
       document.getElementById('tuloste1').innerHTML =
       document.getElementById('tuloste1').innerHTML + " " +
       document.getElementById('luku2').value;
       document.getElementById('luku1').style.visibility =
       'hidden';
       document.getElementById('luku2').style.visibility =
       'hidden';
       }
   if (vaihe > 6) {
       document.getElementById('tuloste2').innerHTML =
       "Anna kaksi desimaalilukua: ";
       if (vaihe == 7) {
          document.getElementById('desi1').style.visibility =
          'visible';
          document.getElementById('desi2').style.visibility =
          'visible';
          }
       else {
            document.getElementById('tuloste2').innerHTML =
            document.getElementById('tuloste2').innerHTML + " " +
            document.getElementById('desi1').value;
            document.getElementById('tuloste2').innerHTML =
            document.getElementById('tuloste2').innerHTML + " " +
            document.getElementById('desi2').value;
            document.getElementById('desi1').style.visibility =
            'hidden';
            document.getElementById('desi2').style.visibility =
            'hidden';
            }
       }

    else {
         document.getElementById('tuloste2').innerHTML = "";
         document.getElementById('desi1').value = "";
         document.getElementById('desi2').value = "";
         document.getElementById('desi1').style.visibility =
         'hidden';
         document.getElementById('desi2').style.visibility =
         'hidden';
         }
    if (vaihe > 10) {
       var luku1 = parseInt
       (document.getElementById("luku1").value, 10);
       var luku2 = parseInt
       (document.getElementById("luku2").value, 10);
```

```
            var jako1 = luku1 / luku2;
            var jako1 = Math.round(jako1);
            // Pöyristetään kahden kokonaisluvun
            // jakolaskun tulos myös kokonaisluvuksi.
            document.getElementById('tuloste3').innerHTML =
            "<br>Jakolaskun tulokset:<br>" + luku1 + " / " +
            luku2 + " = ";
            document.getElementById('tuloste3').innerHTML =
            document.getElementById('tuloste3').innerHTML +
            jako1 + "<br>";
            if (vaihe > 11) {
               var desi1 = parseFloat
               (document.getElementById("desi1").value, 10);
               var desi2 = parseFloat
               (document.getElementById("desi2").value, 10);
               var jako2 = desi1 / desi2;
               document.getElementById('tuloste3').innerHTML =
               document.getElementById('tuloste3').innerHTML +
               document.getElementById("desi1").value + " / " +
               document.getElementById("desi2").value + " = ";
               document.getElementById('tuloste3').innerHTML =
               document.getElementById('tuloste3').innerHTML + jako2;
               }
            }
         else document.getElementById('tuloste3').innerHTML = "";
         }
      else {
            document.getElementById('tuloste1').innerHTML = "";
            document.getElementById('tuloste2').innerHTML = "";
            document.getElementById('tuloste3').innerHTML = "";
            document.getElementById('luku1').value = "";
            document.getElementById('luku2').value = "";
            document.getElementById('desi1').value = "";
            document.getElementById('desi2').value = "";
            document.getElementById('luku1').style.visibility =
            'hidden';
            document.getElementById('luku2').style.visibility =
            'hidden';
            document.getElementById('desi1').style.visibility =
            'hidden';
            document.getElementById('desi2').style.visibility =
            'hidden';
            }
      // Seuraava if-else -rakenne määrittää
      // näkymän muistin sisällöstä.
      if (vaihe > 2) {
            document.getElementById("muisti").innerHTML = "luku1 = ";
            if (vaihe > 5) {
               if (vaihe == 6)
                  document.getElementById("muisti").innerHTML =
                  document.getElementById("muisti").innerHTML +
                  "<span style='color:FF6600;'>" +
                  document.getElementById('luku1').value;
               else
                  document.getElementById("muisti").innerHTML =
                  document.getElementById("muisti").innerHTML +
                  document.getElementById('luku1').value;
               // Muutetaan muuttujan arvon väri oranssiksi, jos
               // se on saanut arvonsa juuri tässä vaiheessa.
               }

            document.getElementById("muisti").innerHTML =
            document.getElementById("muisti").innerHTML +
            "<br>luku2 = ";
```

```
if (vaihe > 5) {
   if (vaihe == 6)
      document.getElementById("muisti").innerHTML =
      document.getElementById("muisti").innerHTML +
      "<span style='color:FF6600;'>" +
      document.getElementById('luku2').value;
   else
      document.getElementById("muisti").innerHTML =
      document.getElementById("muisti").innerHTML +
      document.getElementById('luku2').value;
   // Muutetaan muuttujan arvon väri oranssiksi, jos
   // se on saanut arvonsa juuri tässä vaiheessa.
   }

document.getElementById("muisti").innerHTML =
document.getElementById("muisti").innerHTML +
"<br>jako1 = ";

if (vaihe > 8) {
   var arvo1 = parseInt
   (document.getElementById("luku1").value, 10);
   var arvo2 = parseInt
   (document.getElementById("luku2").value, 10);
   var jako = arvo1 / arvo2;
   var jako = Math.round(jako);
   // Pöyristetään kahden kokonaisluvun jakolaskun
   // tulos myös kokonaisluvuksi.
   if (vaihe == 9)
      document.getElementById("muisti").innerHTML =
      document.getElementById("muisti").innerHTML +
      "<span style='color:FF6600;'>" + jako;
   else
      document.getElementById("muisti").innerHTML =
      document.getElementById("muisti").innerHTML + jako;
   }

document.getElementById("muisti").innerHTML =
document.getElementById("muisti").innerHTML + "<br>";

if (vaihe > 3) {
   document.getElementById("muisti").innerHTML =
   document.getElementById("muisti").innerHTML +
   "desi1 = ";
if (vaihe > 7) {
   if (vaihe == 8)
      document.getElementById("muisti").innerHTML =
      document.getElementById("muisti").innerHTML +
      "<span style='color:FF6600;'>" +
      document.getElementById('desi1').value;
   else
      document.getElementById("muisti").innerHTML =
      document.getElementById("muisti").innerHTML +
      document.getElementById('desi1').value;
      // Muutetaan muuttujan arvon väri oranssiksi,
      // jos se on saanut arvonsa juuri tässä vaiheessa.
   }
document.getElementById("muisti").innerHTML =
document.getElementById("muisti").innerHTML +
"<br>desi2 = ";
if (vaihe > 7) {
   if (vaihe == 8)
      document.getElementById("muisti").innerHTML =
      document.getElementById("muisti").innerHTML +
```

```
            "<span style='color:FF6600;'>" +
            document.getElementById('desi2').value;
        else
            document.getElementById("muisti").innerHTML =
            document.getElementById("muisti").innerHTML +
            document.getElementById('desi2').value;
            // Muutetaan muuttujan arvon väri oranssiksi
            // jos se on saanut arvonsa juuri tässä vaiheessa.
            }
        document.getElementById("muisti").innerHTML =
        document.getElementById("muisti").innerHTML +
        "<br>jako2 = ";
        if (vaihe > 9) {
            var arvo1 = parseFloat
             (document.getElementById("desi1").value, 10);
            var arvo2 = parseFloat
             (document.getElementById("desi2").value, 10);
             var jako = arvo1 / arvo2;
             if (vaihe == 10)
               document.getElementById("muisti").innerHTML =
               document.getElementById("muisti").innerHTML +
               "<span style='color:FF6600;'>" + jako;
             else
               document.getElementById("muisti").innerHTML =
               document.getElementById("muisti").innerHTML +
               jako;
             }
        }
    }
else document.getElementById("muisti").innerHTML = "";
// Seuraava if-else -rakenne määrittää selostuslauseen.
if (vaihe == 1) {
     document.getElementById("selostus").innerHTML =
     "Ohjelmaan sisällytetään kirjasto, jossa " +
     "on syöttö ja tulostusfunktiot.";
     }
else if (vaihe == 2) {
         document.getElementById("selostus").innerHTML =
         "Pääohjelma alkaa.";
         }
else if (vaihe == 3 || vaihe == 4) {
         document.getElementById("selostus").innerHTML =
         "Luodaan pääohjelman muuttujia.";
         }
else if (vaihe == 5 || vaihe == 7) {
         document.getElementById("selostus").innerHTML =
         "Ohjelma tulostaa tekstiä.";
         }
else if (vaihe == 6 || vaihe == 8) {
         document.getElementById("selostus").innerHTML =
         "Ohjelma lukee käyttäjän syöttämät arvot ja" +
          "tallentaa ne muuttujiin.";
          }
 else if (vaihe == 9) {
          document.getElementById("selostus").innerHTML =
          "Jakolaskun tulos sijoitetaan muuttujaan jako1.";
          }
 else if (vaihe == 10) {
          document.getElementById("selostus").innerHTML =
          "Jakolaskun tulos sijoitetaan muuttujaan jako2.";
          }
 else if (vaihe == 11 || vaihe == 12) {
          document.getElementById("selostus").innerHTML =
          "Ohjelma tulostaa tekstin sekä muuttujien arvot.";
```

```
                }
        else if (vaihe == 13) {
                document.getElementById("selostus").innerHTML =
                "Ohjelma päättyy ja palauttaa kokonaisluvun 0.";
                }
}
</script>
</head>
<table width="100%" cellpadding="0" cellspacing="10" border="0">
<tr>
<td width="50%" valign="top">
<font face="arial" size="4">Ohjelman koodi</font>
<!-- Alla olevan div-osio näyttää millaista C++ -esimerkkikoodia käsi-
tellään. Esim. span id="1" tai div id="3" määrittävät rajan joka näy-
tetään eri värillä merkittynä, kun koodia ollaan käsittelemässä samai-
sessa vaiheessa. -->
<div id="koodi" style="height:490px;width:420px;border:1px solid #ccc;
Serif;overflow:auto;">
<font face="courier" size="2">
<span style="color: blue;">
<i>// ohjelma tulostaa annettujen lukujen osamäärän</i><br>
<i>// desimaaliluku syötettävä desimaaliPILKUN
kanssa</i><br><br></span>
<span style="color: green;">
<span id="1">
#include <text><</text>iostream.h<text>></text><br><br>
</span>
</span>
<div id="2">
<b>int</b> main()
</div>
{
<div id="3">
 <b>int</b> luku1, luku2, jako1;
</div>
<div id="4">
 <b>double</b> desi1, desi2, jako2;
</div>
<div id="5">
 cout << <span style="color: red;">"Anna kaksi kokonaislukua:
"</span>;
</div>
<div id="6">
 cin >> luku1 >> luku2;<br>
</div>
<div id="7">
 cout << <span style="color: red;">"Anna kaksi desimaalilukua:
"</span>;
</div>
<div id="8">
 cin >> desi1 >> desi2;
</div>
<div id="9">
 jako1 = luku1 / luku2;
</div>
<div id="10">
 jako2 = desi1 / desi2;
</div>
<div id="11">
 cout << <span style="color: red;">"Jakolaskun tulokset:\n
"</span> << luku1<br>
<span style="padding: 0 20px"> </span><< <span style="color:
red;">" / "</span>
<< luku2 << <span style="color: red;">" = "</span> << jako1 << endl;
```

```
</div>
<span id="12">
 cout << desi1 << <span style="color: red;">" / "</span> << desi2
<< <span style="color: red;">" = "</span><br>
<span style="padding: 0 20px"> </span><< jako2 << endl;<br><br>
</span>
<div id="13">
 <b>return</b> <span style="color: purple;">0</span>;<br>
</div>
}
</font>
</div>
</td>
<td width="50%" valign="top">
<font face="arial" size="4">Näyttö</font>
<font face="courier" size="2">
<!-- Alla olevan div-osion olisi tarkoitus näyttää esimerkkiohjelman
tulostukset konsoliin. -->
<div id="naytto" style="height:103px;width:420px;border:1px solid
#ccc; Serif;overflow:auto;">
<span id="tuloste1" >
<!-- Tulostetaan teksti "Anna kaksi kokonaislukua: ", kun ollaan tie-
tyssä vaiheessa. -->
</span>
<input type="number" id="luku1" class="inputSmall"
style="visibility:hidden"/>
<input type="number" id="luku2" class="inputSmall"
style="visibility:hidden"/>
<br>
<span id="tuloste2">
<!-- Tulostetaan teksti "Anna kaksi desimaalilukua: ", kun ollaan tie-
tyssä vaiheessa. -->
</span>
<input type="number" id="desi1" class="inputSmall"
style="visibility:hidden"/>
<input type="number" id="desi2" class="inputSmall"
style="visibility:hidden"/>
<span id="tuloste3">
</span>
</div>
<!-- Tulostetaan jakolaskujen tulokset,
kun ollaan tietyssä vaiheessa. -->
<br>
<font face="arial" size="4">Muisti</font>
<!-- Alla olevan div-osion olisi tarkoitus näyttää ohjelman muistiin
tallentamat yksityiskohdat, kuten muuttujien arvot. -->
<div id="muisti" style="height:206px;width:420px;border:1px solid
#ccc; Serif;overflow:auto;"></div>
<br>
<font face="arial" size="4">Ehdot</font>
<!-- Alla olevan div-osion olisi tarkoitus näyttää koodissa esiintyvi-
en if-, else-, switch-, do-, for- ja while-ehtolauseiden totuusarvot.
-->
<div id="ehdot" style="height:103px;width:420px;border:1px solid #ccc;
Serif;overflow:auto;">
</div>
</font>
</td>
</tr>
</table>
<span style="color: red;" size="2">
<div id="selostus">
Aloita ohjelman suoritus painamalla Seuraava-painiketta.
</div>
```

```
</span>
<br>
<button id="restartButton" onClick="paivita(0)">Alkuun</button>
<button id="prevButton" class="Buttons"
onClick="paivita(1)">Edellinen</button>
<button id="nextButton" class="Buttons"
onClick="paivita(2)">Seuraava</button>
</html>
```

Liite 2. Kuvan 3 harjoitussimulaation HTML-koodi

```
<html>
<script src="../simulaatio.js" type="text/javascript"></script>
<!-- Mahdollistaa simulaatio.js -tiedoston JavaScript-
funktioiden tarkista() ja uusiTeht() käytön. -->
<head>
<meta charset="utf-8">
<title>Harjoitus</title>

<link rel="stylesheet" type="text/css" href="../sheets.css">
<script>
function arvonta() {
    tehtava[0] = uusiTeht(1, 3);
    if (tehtava[0] == 1) document.getElementById("ohje").innerHTML =
           "Luku1 oli suurempi.";
    else if (tehtava[0] == 2)
           document.getElementById("ohje").innerHTML =
           "Luku2 oli suurempi.";
    else if (tehtava[0] == 3)
           document.getElementById("ohje").innerHTML =
           "Luvut olivat yhta suuret.";
           document.getElementById("naytto").innerHTML = "";
           document.getElementById('luku1').value = "";
           document.getElementById('luku2').value = "";
}
function ratkaisu() {
    if (document.getElementById('luku1').value == "" ||
    document.getElementById('luku2').value == "") {
           alert("Anna kokonaislukuina alkuarvot
           muuttujille luku1 ja luku2.");
           }
    else {
           var int1 =
           parseInt(document.getElementById("luku1").value, 10);
           var int2 =
           parseInt(document.getElementById("luku2").value, 10);
           var float1 =
           parseFloat(document.getElementById("luku1").value, 10);
           var float2 =
           parseFloat(document.getElementById("luku2").value, 10);

    if (int1 != float1 || int2 != float2) {
           alert("Ohjelma kaatuu, jos syötteenä annetaan jotakin muuta
           kuin kokonaisluku. Korjaa virheelliset syötteet.");
           }
    else {
           document.getElementById("palaute").innerHTML = "";
           if (int1 > int2)
             document.getElementById("naytto").innerHTML =
             "Luku1 oli suurempi.";
           else if (int2 > int1)
                document.getElementById("naytto").innerHTML =
                "Luku2 oli suurempi.";

           else if (int1 == int2)
                document.getElementById("naytto").innerHTML =
                "Luvut olivat yhta suuret.";
           if (tehtava[0] == 1) {
              if (int1 > int2)
                 tarkista(true,
```

```
                    "Annoit oikeat arvot muuttujille luku1 ja luku2.");
                else
                    tarkista(false,
                    "Muuttujan luku1 täytyy olla suurempi kuin luku2.");
                }
            else if (tehtava[0] == 2) {
                    if (int2 > int1) tarkista(true, "Annoit oikeat arvot
                    muuttujille luku1 ja luku2.");
                    else tarkista(false, "Muuttujan luku2 täytyy olla
                    suurempi kuin luku1.");
                    }
            else if (tehtava[0] == 3) {
                    if (int1 == int2)
                        tarkista(true, "Annoit oikeat arvot
                        muuttujille luku1 ja luku2.");
                    else
                        tarkista(false, "Muuttujien täytyy olla
                        yhtä suuria.");
                    }
            }
    }
}
</script>
</head>
<table width="100%" cellpadding="0" cellspacing="10" border="0">
<tr>
<td width="50%" valign="top">
<font face="arial" size="4">Ohjelman koodi</font>
<div style="height:490px;width:420px;border:1px solid #ccc; Ser-
if;overflow:auto;">
<font face="courier" size="2">
<span style="color: green;">
#include <text><</text>iostream.h<text>></text><br><br>
</span>
<b>int</b> main()<br>
{<br>
 <b>int</b> luku1 = <input type="number" id="luku1"
class="inputSmall" />,
luku2 = <input type="number" id="luku2" class="inputSmall" />;<br>
 <b>if</b> (luku1 > luku2)<br>
  <b>cout</b> << <span style="color: red;">"Luku1 oli
suurempi."</span>;<br>
 <b>else if</b> (luku2 > luku1)<br>
  <b>cout</b> << <span style="color: red;">"Luku2 oli
suurempi."</span>;<br>
 <b>else if</b> (luku1 == luku2)<br>
  <b>cout</b> << <span style="color: red;">"Luvut olivat
yhta suuret."</span>;<br><br>
 <b>return</b> <span style="color: purple;">0</span>;<br>
}</font>
</div>
</td>
<td width="50%" valign="top">
<font face="arial" size="4">Ohjelman tuloste</font>
<font face="courier" size="2">
<div id="naytto" style="height:103px;width:420px;border:1px solid
#ccc; Serif;overflow:auto;">
</div></font><br>
<font face="arial" size="4">Tehtävä</font>
<div id="tehtava" style="height:103px;width:420px;border:1px solid
#ccc; Serif;overflow:auto;">
<span style="color: purple;">
Anna oheiseen ohjelmakoodiin sellaiset alkuarvot muuttujille<br>
luku1 ja luku2, että ohjelman tulostus on seuraava:</span>
```

```
<font face="courier" size="2">
<div style="color: black;" id="ohje">
Luku2 oli suurempi.
</div></font>
</div><br>
<script>
tehtava[0] = 2;
</script>
<font face="arial" size="4">Palaute</font>
<div id="palaute" style="height:195px;width:420px;border:1px solid
#ccc; Serif;overflow:auto;"></div>
</td>
</tr>
</table>
<button id="checkButton" class="Buttons"
onClick="ratkaisu()">Tarkista</button>
<button id="newButton" class="Buttons"
onClick="arvonta()">Uusi</button>
<br></br>
</html>
```

Liite 3. Kuvien 25–27 simulaatioiden vastine esimerkin 2 tapaukselle

```
if (vaihe == 10) {
   var potenssi = parseInt(document.getElementById('potenssi').value);
   edvaih = vaihe - 1;
   if (potenssi < 1) seurvaih = 21;
   else seurvaih = vaihe + 1;
   }
else if (vaihe == 12) {
        var potenssi = parseInt
           (document.getElementById('potenssi').value);
        if (toisto[0] < (potenssi - 1)) edvaih = 17;
        else edvaih = vaihe - 1;
        seurvaih = vaihe + 1;
        }
else if (vaihe == 14) {
        edvaih = vaihe - 1;
        if (toisto[0] == 0) seurvaih = vaihe + 1;
        else seurvaih = 16;
        }
else if (vaihe == 15) {
        edvaih = vaihe - 1;
        var potenssi = parseInt
           (document.getElementById('potenssi').value);
        if (toisto[1] < potenssi) seurvaih = 18;
        else seurvaih = 19;
        }
else if (vaihe == 16) {
        edvaih = 14;
        seurvaih = vaihe + 1;
        }
else if (vaihe == 17) {
        edvaih = vaihe - 1;
        if (toisto[0] > 0) seurvaih = 12;
        else seurvaih = 18;
        }
else if (vaihe == 18) {
        if (toisto[1] > 2) edvaih = vaihe;
        else if (toisto[1] == 2) edvaih = 15;
        else edvaih = vaihe - 1;
        var potenssi = parseInt
           (document.getElementById('potenssi').value);
        if (toisto[1] < potenssi) seurvaih = vaihe;
        else seurvaih = vaihe + 1;
        }
else if (vaihe == 19) {
        if (toisto[1] == 1) edvaih = 15;
        else edvaih = vaihe - 1;
        seurvaih = vaihe + 1;
        }
else if (vaihe == 20) {
        edvaih = vaihe - 1;
        seurvaih = 23;
        }
else if (vaihe == 21) {
        edvaih = 10;
        seurvaih = vaihe + 1;
        }
```

```
else if (vaihe == 23) {
        var potenssi = parseInt
           (document.getElementById('potenssi').value);
        if (potenssi < 1) edvaih = vaihe - 1;
        else edvaih = 20;
        seurvaih = vaihe;
        }
else {
     edvaih = vaihe - 1;
     seurvaih = vaihe + 1;
     }
...
// Seuraava if-else -rakenne määrittää mihin vaiheeseen siirrytään
// sekä sen kuinka monetta kertaa toistetaan samaa aliohjelmaa.
if (toiminto == 0) alkuun();
else if (toiminto == 1) {
        if (vaihe == 12) toisto[0]++;
        else if (vaihe == 15 || vaihe == 18) toisto[1]--;
        edvaihe(edvaih);
        if (vaihe == 10) toisto[0] = 5;
        }
else if (toiminto == 2) {
        if (vaihe == 5) {
           if (document.getElementById('luku').value == "") {
              alert("Ohjelma pysähtyi odottamaan syötettä, eikä etene
              ennen kuin olet syöttänyt kokonaisluvun.");
              }
           else {
               var luku = parseInt
                  (document.getElementById('luku').value, 10);
               document.getElementById('luku').value =
               luku.toString();
               seurvaihe(seurvaih);
               }
           }
        else if (vaihe == 7) {
                if (document.getElementById('potenssi').value == "") {
                   alert("Ohjelma pysähtyi odottamaan syötettä, eikä
                   etene ennen kuin olet syöttänyt kokonaisluvun.");
                   }
                else {
                    var potenssi = parseInt
                       (document.getElementById('potenssi').value, 10);
                    if (potenssi < 6) {
                       document.getElementById('potenssi').value =
                       potenssi.toString();
                       seurvaihe(seurvaih);
                       }
                    else
                       alert("Voit antaa vain luvun, joka on
                       pienempi tai yhtä suuri kuin 5.");
                    }
                }

        else {
             seurvaihe(seurvaih);
             if (vaihe == 11) toisto[0] = parseInt
                 (document.getElementById('potenssi').value);
             else if (vaihe == 12) toisto[0]--;
             else if (vaihe == 15 || vaihe == 18) toisto[1]++;
             }
        }
```