OHJ-1150 Ohjelmointi II – tentti 06.02.2012

tentin laatinut: Ari Suntioinen *<ari.suntioinen@tut.fi>*

Tentissä saa olla esillä mitä tahansa kirjallista materiaalia (siis tekstiä ja kuvia paperilla). Elektronista materiaalia, mukaanlukien laskimet, ei saa olla esillä.

Tehtävä 1

Kirjoita ***rekursiivinen*** funktio, joka saa parametreinaan

- `list<string>::const_iterator` -tyyppisen iteraattorin,
- merkkijonon (`string`) ja
- jotain muuta tarpeellista.

Funktion on tarkoitus palauttaa `int`-tyyppinen arvo, joka kertoo, kuinka monta kertaa parametrina annettu merkkijono esiintyy parametrina annetun iteraattorin määräämässä listassa. Esimerkiksi, jos parametrimerkkijono olisi `"abc"` ja iteraattorin osoittama lista sisältäisi järjestyksessä alkiot:

```
{ "ab", "abc", "a", "", "abc", "abcd", "abc" }
```

paluuarvo olisi 3.

Esitä myös kuinka funktiotasi kutsuttaisiin.

Vastaa perustellen, onko toteuttamasi funktio häntärekursiivinen vai ei? [25 p]

Tehtävä 2

Tämä tehtävä ei varsinaisesti ole "esseetehtävä", joten ei kannata välttämättä lähteä sivutolkulla selostamaan. Riittää kun asiat selvittää lyhyesti, mutta kuitenkin kokonaisia virkkeitä käyttäen. [25 p]

(a) Esittele lyhyesti, mitä valmiita mekanismeja C++ (ja ohjelmointikielet yleisestikin ottaen) tarjoavat ohjelman monimutkaisuuden hallintaan "hajoita ja hallitse" -periaateen mukaisesti, eli siis siten, että vaikeita tai suuria kokonaisuuksia voidaan jakaa pienempiin selkeisiin osiin. Osaatko lisäksi kertoa, mikä näille kaikille tavoille on yhteistä?

(b) Kerro lyhyesti, mitä yhteistä ja eroa taulukoilla ja osoittimilla on. Mitä on osoitinaritmetiikka ja mikä on sen yhteys tähän kaikkeen?

Käännä ➠

Tehtävä 3

Ajatellaan seuraavanlaisista dynaamisesti varatuista `Alkio`-structeista:

```
struct Alkio {
    int     luku;
    string  tunniste;
    Alkio  *seur;
    Alkio  *syv;
};
```

koostuvaa linkitettyä rakennetta, jonka ensimmäisen alkion sijainnista muistissa pidetään kirjaa osoittimella `eka`:

```
Alkio *eka = nullptr;
```

Rakenteelle on määritelty lisäysalgoritmi (tilanteen yksinkertaistamiseksi koodissa on oletettu, ettei muistinvaraus epäonnistu):

```
Alkio *uusi = new Alkio;
uusi->luku = lisattava;
uusi->tunniste = sana;
uusi->seur = uusi->syv = nullptr;
Alkio *apu = eka;
while ( apu != nullptr ) {
    if ( apu->luku == lisattava ) {
        break;
    }
    apu = apu->seur;
}
if ( apu == nullptr ) {
    uusi->seur = eka;
    eka = uusi;
} else {
    uusi->seur = apu->syv;
    uusi->syv = apu;
    apu->syv = uusi;
}
```

Edellisen algoritmin huomioiden:

(a) Piirrä laatikko–nuoli-kaaviona tilanne, kun tyhjään rakenteeseen on lisätty järjestyksessä luku-tunniste-parit: `{1, "a"}`, `{2, "b"}`, `{3, "c"}`, `{3, "d"}`, `{4, "e"}`, `{2, "f"}`, `{2, "g"}` ja `{1, "h"}`.

(b) Esitä edellä olleen koodinpätkän tyylisellä algoritmilla, kuinka tulostaisit näytölle kaikkien rakenteessa olevien solujen `luku`-kentät.

(c) Esitä vielä koodinpätkä sille, kuinka poistaisit rakenteesta kaikki tietyn kokonaisluvun sisältävät solut (esim. kaikki kolmoset). [25 p]

Tehtävä 4

Lue tehtävä ensin kokonaisuudessaan läpi: valitsemastasi toteutustavasta riippuen myöhemmillä kohdilla saattaa olla vaikutusta aiempiin kohtiin.

Pohditaan `Makefile`ä. Aihepiiri on kaikille tuttu kurssin tässä vaiheessa. Keskitytään tarkastelussa puhtaasti riippuvuussuhteiden esittämiseen ja unohdetaan päivityskomennot, makromäärittelyt yms.

`Makefile` kuvailee tiedostojen välisiä riippuvuuksia tekstuaalisessa muodossa. Esimerkiksi rivit:

```
tiedostoA: tiedostoB tiedostoC
tiedostoC: tiedostoB tiedostoD
```

kertovat, että `tiedostoA` riippuu tiedostoista `tiedostoB` ja `tiedostoC`, ja `tiedostoC` riippuu tiedostoista `tiedostoB` ja `tiedostoD`. Käytännössä tämä tarkoittaa sitä, että jos jompaa kumpaa tai molempia tiedostoista `tiedostoB` tai `tiedostoC` muutetaan, `tiedostoA` ei enää sen jälkeen ole ajan tasalla. Toisaalta myös muutos `tiedostoD`:ssä johtaa `tiedostoA`:n vanhenemiseen, koska se riippuu `tiedostoD`:stä epäsuoraan `tiedostoC`:n välityksellä. Lyhyesti sanottuna: `Makefile` kuvaa tiedostojen riippuvuussuhteiden verkon.

Suunnittele tietotyyppi, jota voidaan käyttää em. riippuvuussuhdeverkkojen käsittelyyn c++-ohjelmassa. Ainakin seuraavat asiat pitää huomioida:

(a) Tyypin julkisen rajapinnan määrittely, siis luokan public-osa.

(b) Missä muodossa riippuvuudet tallennetaan private-osaan (a- ja b-kohdat voi yhdistää yhdeksi luokkamäärittelyksi).

(c) c++-toteutus funktiolle, jonka avulla rakenteeseen voidaan lisätä yhden tiedoston riippuvuustiedot (tiedoston nimi ja mistä tiedostoista se riippuu suoraan).

(d) c++-toteutus funktiolle, jonka ***paluuarvona*** saadaan kaikki tiedostot, joista parametritiedosto riippuu suoraan. Aiempaa esimerkkiä soveltaen: jos tutkittava tiedosto on `tiedostoA`, tämä funktio palauttaisi `tiedostoB` ja `tiedostoC`, eli tiedostot, jotka `Makefile`:ssä on merkitty suoriksi riippuvuuksiksi.

(e) Toteuta c++-funktio, jonka paluuarvo on `true`, vain jos ensimmäisenä parametrina annettu tiedosto riippuu suoraan tai epäsuoraan toisena parametrina annetusta tiedostosta. Esimerkiksi paluuarvo olisi `true`, jos funtiolle annettaisiin parametreina `tiedostoA` ja `tiedostoD`, tai paluuarvo olisi `false`, jos parametrit olisivat `tiedostoB` ja `tiedostoD`.

Tämä ***tehtävä on tarkoitus toteuttaa luokkien ja STL-säiliöiden avulla.*** Muunlaisista ratkaisuista saatu pistemäärä lähestyy asymptoottisesti nollaa, kun koodirivien määrä lähestyy ääretöntä.

Huomaa, että ***vastauksissa ei saa näkyä sanoja*** `cin`, `cout` tai `cerr`. Pääohjelmaa `main` ei myöskään ole tarkoitus toteuttaa. [25 p]