Demo 6 vastauksia

1. tehtävä

```
#ifndef __D6T1_H__
#define __D6T1_H__

#include <iostream>
using std::ostream;
using std::cout;
using std::endl;

#include <string>
using std::string;                                                    10

template <typename T>
class Rationaaliluku {
        T osoittaja;
        T nimittaja;
        bool sievennettava;
public:
        Rationaaliluku(T osoittajaksi,
                       T nimittajaksi,
                       bool sievennetaanko=false)                     20
        : osoittaja(osoittajaksi), nimittaja(nimittajaksi),
          sievennettava(sievennetaanko) {
                if (nimittaja==0)
                        throw new Nollanimittaja(osoittajaksi, nimittajaksi);
        }

        Rationaaliluku(const Rationaaliluku& toinen)
        : osoittaja(toinen.osoittaja), nimittaja(toinen.nimittaja) {}

        Rationaaliluku& operator *=(const Rationaaliluku& toinen) {   30
                osoittaja*=toinen.osoittaja;
                nimittaja*=toinen.nimittaja;
                return *this;
        }

        Rationaaliluku& operator /=(const Rationaaliluku& toinen) {
                osoittaja*=toinen.nimittaja;
                nimittaja*=toinen.osoittaja;
                return *this;
        }                                                             40

        Rationaaliluku& operator +=(const Rationaaliluku& toinen) {
                if (nimittaja==toinen.nimittaja) {
                        osoittaja+=toinen.osoittaja;
                        return *this;
```

```
                } else
                        throw string("Laiska ope ei jaksanu"
                                "tehä kunnollista yhteenlaskua, hyi!");
        }
        void sievenna() {}

        ostream& tulosta(ostream& out) {
                out << osoittaja << "/" << nimittaja;
                return out;
        }

        struct Nollanimittaja {
                Nollanimittaja(T o, T n) : osoittaja(o), nimittaja(n) {};
                T osoittaja;
                T nimittaja;
        };
};

template <typename T>
ostream& operator << (ostream& out, Rationaaliluku<T> r) {
        return r.tulosta(out);
}

template <typename T>
Rationaaliluku<T> operator *(const Rationaaliluku<T>& a,
                             const Rationaaliluku<T>& b) {
        Rationaaliluku<T> c(a);
        return c*=b;
}

template <typename T>
Rationaaliluku<T> operator /(const Rationaaliluku<T>& a,
                             const Rationaaliluku<T>& b) {
        Rationaaliluku<T> c(a);
        return c/=b;
}

template <typename T>
Rationaaliluku<T> operator +(const Rationaaliluku<T>& a,
                             const Rationaaliluku<T>& b) {
        Rationaaliluku<T> c(a);
        return c+=b;
}

template <>
Rationaaliluku<int>&
Rationaaliluku<int>::operator *= (const Rationaaliluku<int>& toinen) {
        cout << " <\"tehostettu\" kertolasku> ";
        osoittaja*=toinen.osoittaja;
```

```
        nimittaja*=toinen.nimittaja;
        return *this;
}

#endif
```

2. tehtävä

```
#ifndef __D6T2_H__
#define __D6T2_H__

template<typename T>
class Sailio {
public:
        Sailio(int maxiksi=3) : max(maxiksi), lkm(0) {
                data=new (T*)[max];
        }
        ~Sailio() {
                delete[] data;
        }

        void lisaa(T *t) {
                if (lkm>=max) {
                        max=max*3/2;
                        T **uusi_data=new (T*)[max];
                        for (int i=0; i<lkm; ++i)
                                uusi_data[i]=data[i];
                        delete[] data;
                        data=uusi_data;
                }
                data[lkm++]=t;
        }

        T* anna(int i) {
                return data[i];
        }

        void poista(int i) {
                for (int j=i+1; j<lkm; ++j)
                        data[j-1]=data[j];
                --lkm;
        }

        int alkioita() const { return lkm; }

        class Sailio_iteraattori {
                Sailio<T>* sailo;
```

```
        int indeksi;
    public:
        Sailio_iteraattori(Sailio<T>* s) : sailo(s), indeksi(0) {}
        bool jaljellako() { return indeksi<sailo->alkioita(); }
        T* nykyinen() { return sailo->anna(indeksi); }
        void seuraavaan() { indeksi++; }
    };

    Sailio_iteraattori iteraattori() {
        return Sailio_iteraattori(this);
    }


private:
    T **data;
    int lkm;
    int max;
};

#endif
```

```
#include <iostream>
using std::cout;
using std::endl;
using std::ostream;

#include <string>
using std::string;

#include "d6t1.h"
#include "d6t2.h"

class Tieto {
public:
    Tieto() : tieto(0) {};
    Tieto(int t) : tieto(t) {
        cout << " Tieto luotu osoitteeseen " << this
             << " arvolla " << tieto << "." << endl;
    };

    // STL vaatii
    Tieto(const Tieto& toinen) : tieto(toinen.tieto) {
        cout << " Kopioitu Tieto-olio osoitteeseen " << this
             << " arvolla " << tieto << " oliosta " << &toinen
             << "." << endl;
    }
```

```
        virtual ~Tieto() {
                cout << " Tietoalkio arvolla " << tieto
                     << " tuhottu." << endl;
        }
                                                                    30
        // STL vaatii
        bool operator < (const Tieto& toinen) {
                return tieto<toinen.tieto;
        }

        // STL vaatii
        Tieto& operator = (const Tieto& toinen) {
                if (&toinen!=this)
                        tieto=toinen.tieto;
                return *this;                                       40
        }

        virtual
        string tyyppi() { return "Tieto"; }

        int tieto;
};

int vector_main(void);
                                                                    50

int main(void) {
        cout << "\n===============================================" << endl;
        cout << "          main" << endl;
        cout << "===============================================" << endl;

        cout << "\nLuodaan Sailio ja kaksi alkiota:" << endl;
        typedef Sailio<Tieto> Tietosailio;
        typedef Tietosailio::Sailio_iteraattori Tietoiteraattori;
                                                                    60
        Tietosailio s(2);
        Tieto *t1=new Tieto(23);
        Tieto *t2=new Tieto(175);

        cout << "\nLisätään kaksi alkiota:" << endl;
        s.lisaa(t1);
        s.lisaa(t2);

        Tietoiteraattori ti=s.iteraattori();
        while (ti.jaljellako()) {                                   70
                Tieto *t=ti.nykyinen();
                cout << "  Tieto: " << t->tieto
                     << " Tyyppi: " << t->tyyppi() << endl;
                ti.seuraavaan();
```

```
        }

        cout << "\nLisätään alkio:" << endl;
        Tieto *t3=new Tieto(42);
        s.lisaa(t3);
        for (Tietoiteraattori i=s.iteraattori(); i.jaljellako(); i.seuraavaan()) {
                Tieto *t=i.nykyinen();
                cout << "  Tieto: " << t->tieto
                     << " Tyyppi: " << t->tyyppi() << endl;
        }

        cout << "\nPoistetaan 2. alkio:" << endl;
        s.poista(1);
        for (int i=0; i<s.alkioita(); ++i) {
                Tieto *t=s.anna(i);
                cout << "  Tieto: " << t->tieto
                     << " Tyyppi: " << t->tyyppi() << endl;
        }

        try {
                Rationaaliluku<int> a(2,5), b(3,4), c=a*b;

                cout << c << endl;
        } catch (string& err) {
                cout << err << endl;
        }

        return vector_main();

        //return 0;
}

#include <vector>
#include <algorithm>


int vector_main(void) {
        cout << "\n===============================================" << endl;
        cout << "          vector_main" << endl;
        cout << "===============================================" << endl;

        using std::vector;
        using std::remove;

        cout << "\nLuodaan Sailio ja kaksi alkiota:" << endl;
        typedef vector<Tieto*> Tietosailio;
        typedef Tietosailio::iterator Tietoiteraattori;

        Tietosailio s;
```

```
Tieto *t1=new Tieto(23);
Tieto *t2=new Tieto(175);

cout << "\nLisätään kaksi alkiota:" << endl;
s.push_back(t1);
s.push_back(t2);
                                                                    130
Tietoiteraattori ti=s.begin();
while (ti!=s.end()) {
        Tieto *t=*ti;
        cout << "  Tieto: " << t->tieto
             << " Tyyppi: " << t->tyyppi() << endl;
        ti++;
}

cout << "\nLisätään alkio:" << endl;
Tieto *t3=new Tieto(42);                                            140
s.push_back(t3);
for (Tietoiteraattori i=s.begin(); i!=s.end(); i++) {
        Tieto *t=*i;
        cout << "  Tieto: " << t->tieto
             << " Tyyppi: " << t->tyyppi() << endl;
}

cout << "\nPoistetaan 2. alkio:" << endl;

s.erase(s.begin()+1);                                               150

/*
  Tietoiteraattori uusi_loppu=
        remove(s.begin(), s.end(), t2); // huomaa ero "Sailo-toteutukseen":
                                        // t2, jonka tiedetään olevan toinen alkio
                                        // poistetaan, vaikka se olisi missä
                                        // kohti vektoria tahansa

  Kas, kas. Kun nyt sitten testasin tuon ohjelman vihdoin,
  kun se oli unohtunut ennen demoja :) sori!, niin huomasin,        160
  että tuo remove() jättää jälkeensä niin monta kopiota viimeistä
  alkiota kuin on poistettuja alkioita, eli se ei muuta
  säiliön kokoa (no, mitenkäs se iteraattorin läpi onnistuisikaan?).

  Niinpä tuo s.erase(s.begin()+1) osoittautui sittenkin paremmaksi
  vaihtoehdoksi.
*/

for (Tietoiteraattori i=s.begin(); i!=s.end(); i++) {
        Tieto *t=*i;                                                170
        cout << "  Tieto: " << t->tieto
             << " Tyyppi: " << t->tyyppi() << endl;
```

```
        }

        try {
                Rationaaliluku<int> a(2,5), b(3,4), c=a*b;

                cout << c << endl;
        } catch (string& err) {
                cout << err << endl;
        }

        for (Tietoiteraattori i=s.begin(); i!=s.end(); i++) {
                delete *i;
        }

        s.clear();
        cout << "Valamista tulj!" << endl;

        return 0;
}


/*
================================================
                main
================================================

Luodaan Sailio ja kaksi alkiota:
  Tieto luotu osoitteeseen 0x804e188 arvolla 23.
  Tieto luotu osoitteeseen 0x804e198 arvolla 175.

Lisätään kaksi alkiota:
    Tieto: 23 Tyyppi: Tieto
    Tieto: 175 Tyyppi: Tieto

Lisätään alkio:
  Tieto luotu osoitteeseen 0x804e1a8 arvolla 42.
    Tieto: 23 Tyyppi: Tieto
    Tieto: 175 Tyyppi: Tieto
    Tieto: 42 Tyyppi: Tieto

Poistetaan 2. alkio:
    Tieto: 23 Tyyppi: Tieto
    Tieto: 42 Tyyppi: Tieto
 <"tehostettu" kertolasku> 6/20

================================================
                vector_main
================================================
```

```
Luodaan Sailio ja kaksi alkiota:
  Tieto luotu osoitteeseen 0x804e178 arvolla 23.
  Tieto luotu osoitteeseen 0x804e1c8 arvolla 175.

Lisätään kaksi alkiota:
    Tieto: 23 Tyyppi: Tieto
    Tieto: 175 Tyyppi: Tieto
                                                          230
Lisätään alkio:
  Tieto luotu osoitteeseen 0x804e320 arvolla 42.
    Tieto: 23 Tyyppi: Tieto
    Tieto: 175 Tyyppi: Tieto
    Tieto: 42 Tyyppi: Tieto

Poistetaan 2. alkio:
    Tieto: 23 Tyyppi: Tieto
    Tieto: 42 Tyyppi: Tieto
 <"tehostettu" kertolasku> 6/20                           240
  Tietoalkio arvolla 23 tuhottu.
  Tietoalkio arvolla 42 tuhottu.
Valamista tulj!
*/
```

---

3. Iteraattorilla voidaan ”iteroida”, eli käydä järjestyksessä läpi säliön alkiot. On tapana tehdä erilaisia iteraattoreita erilaisille järjestyksille, esimekiksi tavallisin iteraattori käy alkiot läpi jossain järjestyksessä yhteen suuntaan, kaksisuuntaisella iteraattorilla suuntaa voi vaihtaa.

   Monisteessa iteraattori oli tällainen ”tavallinen”, sillä oli metodit

   `jaljellako()` joka palautti tiedon siitä, onko kaikki säilön alkiot jo käyty läpi,

   `nykyinen()` joka palautti iteraattorin nykyisen, kutsuhetkellä osoittaman alkion ja

   `seuraavaan()` joka siirsi iteraattorin osoittamaan seuraavaa alkiota.

   Iteraattorin muodostaa yleensä säilöluokka, jolta myös iteraattorin saa käyttöönsä.

   STL:n iteraattorikategoriat: input, output, forward, bidirectional ja random access. Iteraattorisovittimia: reverse, insert, stream.

4. `vector`

   - metodeja: front, back, indeksointi ja at, push_back ja pop_back, insert, erase, size, empty, clean, reserve.
   - iteraattori: begin, end ja ++
   - huomaa alkion poisto, list olisi parempi, kenties?

5. Eri säilöluokat soveltuvat eri tarpeisiin:

   `vector` vanha tuttu ”taulukko”

   `list` kuten nimi sanoo, lista, vieläpä kahteen suuntaan linkitetty. Ei mielivaltaisen alkion indeksointia, lisääminen vakioaikaista. splice.

   `deque` on taulukko, jonka molempiin päihin tehtävät lisäykset ja poistot ovat vakioaikaisia, muualle lineaarisia. ”Korttipakka”. Indeksointi hitaampaa kuin vektorilla, alkuun lisääminen vakioaikaista (vektorilla lineaarinen).

   `set` on assosiatiivinen säiliö. Joukko, yksi esiintymä alkiosta mahdollinen. Järjestys avaimen mukaan. Avain itse alkio.

   `multiset` mahdollistaa monta esiintymää alkiosta.

   `map` on kuten set, mutta avain ja alkio ovat eri oliot. Jos indeksointi ei löydä alkiota avaimelle, luodaan alkion oletusrakentimella avaimelle alkio map:iin.

   `multimap` mahdollistaa useita alkioita yhdelle avaimelle.

   `vector<bool>` totuusarvovektori

   `bitset` binaaristen bittisarjojen käsittelyyn, esimerkiksi `bitset<64>`.

   `stack` on *säiliösovitin*, säiliötä voidaan käyttää kuten pinoa.
   `stack<int, list<int>>`

   `queue` jonosovitin

   `priority_queue` kuten jonosovitin, mutta alkiot suuruusjärjestyksessä.

6. Algoritmeja: copy, find, sort, merge(alku1, loppu1, alku2, loppu2, kohde), for_each(alku, loppu, funktio), partition(alku, loppu, ehtofunktio), random_shuffle(alku, loppu).

   Algoritmeja voi iteraattoreiden avulla käyttää osalle säiliön alkioista tai kaikille. Iteraattoreiden avulla funktiot voivat käsitellä erilaisten säiliöiden alkioita säiliöistä riippumatta. Algoritmien toimintaa

voi muuttaa iteraattorisovittimien avulla (find ja backward_iterator). Myös omia iteraattoreita (ja algoritmeja) voi kirjoittaa.

7. tehtävä

Periytyminen	**Delegointi, Koostuminen**
is-a	has-a
...	...