Programmeerimine keeles C++

Praktikum 3: mallid, C++ standardteegi konteinerid, iteraatorid

Ülesanne 1 – Geomeetria suvalise mõõtmega ruumis.

Viimases geomeetriateegiga seotud praktikumis kirjutame seni kõige täiuslikumad vektori, sirglõigu ja kera klassid. Nimelt kasutame klassimalle, et luua geomeetrilisi objekte suvalise mõõtmega ruumis (märkus: ma palun, et teie kood annaks triviaalseid vastuseid ka siis, kui parameetritega määratakse 0-mõõtmeline ruum). Järgmiste klasside realiseerimiseks kasutage malle.

Ärge laske ülesannete pikal tekstil ennast hirmutada. Enamuse lahendused on võrdlemisi lihtsad.

Universaalne vektor (**`vector.h`**)

Looge klassi `Vector`, millele saab ette anda murdarvuliste koordinaatide arvu (`template <unsigned short n>`). Etteantav mittenegatiivne täisarv `n` määrab, mitu mõõdet vektoril on. Vektori koordinaate salvestage STL klassi `std::vector` abil klassi muutujas nimega `coords`.

Mõned näited:

```
Vector<2> kahem66tmelinePunkt;      // punkt tasandil

Vector<10> kymnem66tmelinePunkt;    // punkt kümnemõõtmelises ruumis
```

Lisage klassile meetodid:

Meetod	**Eesmärk**
`Vector<n> ()`	algväärtustab koordinaadid nullidega
`Vector<n> (vector<float> crds)`	väärtustab vektori koordinaadid etteantud väärtustega
`distanceFrom (Vector<n> v)`	tagastab kauguse teisest sama mõõtmehulgaga punktist
`string toString ()`	tagastab vektori esituse sõnena $(x_1, x_2, \ldots, x_n)$

Meetodid `distanceFrom` ja `toString` realiseerige ühel meetodil kahest:

1. teete iteraatori üle koordinaatide vektori, või
2. kasutate standardteegi `for_each` algoritmi ning kirjutate funktsiooniobjekti, mis aitab vastavat operatsiooni läbi viia.

Juhul, kui tekivad veaolukorrad, näiteks kui malliga määratud koordinaatide arv n erineb etteantud koordinaatide vektori suurusest, visake string-tüüpi erind (vt materjali). Erindi teksti kirjutage inimkeelne selgitus selle kohta, mis toimus. Arvestage, et 0-koordinaadiga vektor on põhimõtteliselt täiesti lubatud ning seda veajuhuks lugema ei peaks.

Lahendus on väärt 3 punkti.

Universaalne sirglõik mall (`line.h`)

Looge klassimall `Line`, mille parameetriks on vektori klass `T` (`template <class T>`). Klass esindab sirglõike üle suvalise suurusega vektorite. Klassil on kaks muutujat – `p1` ja `p2`, mõlemad on tüüpi `T` – need esindavad sirglõigu tippe. Tulemusena peab saama teha kahe tipuga sirglõike nii:

```
Line< Vector<2> > kahem66tmelineSirgl6ik;
Line< Vector<7> > seitsmem66tmelineSirgl6ik;
```

NB! Pange tühikud `Vector<n>` ümber, muidu arvab kompilaatoril, et seal on voo operaator `>>`.

Lisage klassile meetodid:

Meetod	**Eesmärk**
`Line<T> ()`	vaikekonstruktor – loob tipud (`T` vaikekonstruktoriga)
`Line<T> (T np1, T np2)`	parameetritega konstruktor - väärtustab klassi elemendid
`float length ()`	tagastab sirglõigu pikkuse kasutades klassi `T` meetodit `distanceFrom`
`string toString ()`	tagastab lõigu esituse sõnena `((tipp1)-(tipp2))`

Lahendus toob 1 punkti.

Universaalne kera (`sphere.h`)

Looge klassimall `Sphere`, mille parameetriks on vektori klass `T` (`template <class T>`). Klass esindab kerasid (ja kahemõõtmelisel erijuhul ringe). Klassil olgu `T` tüüpi muutuja `o`, mis esitab keskpunkti ja `float`-tüüpi murdarv `r`, mis esitab raadiust. Tulemusena peab saama teha ringe ja kerasid nii:

```
Sphere< Vector<2> > ring;
Sphere< Vector<3> > kera;
```

Meetod	**Eesmärk**
`Sphere<T> ()`	vaikekonstruktor – loob `T` tüüpi tipu ja paneb raadiuseks nulli
`Sphere<T> (T no, float nr)`	parameetritega konstruktor – kasutab antud tippu ja raadiust
`bool contains (T v)`	tagastab `true`, kui tipp on kera pinnal või sees, muidu `false`
`bool contains (Line<T> l)`	tagastab `true`, kui antud lõik on kera sees, muidu `false`
`void scale (float factor)`	korrutab kera raadiuse antud väärtusega
`string toString ()`	tagastab lõiku esituse sõnena `((tipp), raadius)`

Lahendus toob 2 punkti.

See ei ole ülesande osa, aga jätke meelde, et keeles on olemas ka võimalus konkreetsete parameetritega implementatsioonide kirjutamiseks. Näiteks võite realiseerida erijuhtudena kahemõõtmelise kera (ringi) jaoks ümbermõõdu ja pindala ning kolmemõõtmelise kera jaoks ruumala arvutamise. Vastav märksõna, mida uurida, on *template specialization*.

NB! Ärge muutke soovitatud muutujanimesid ning hoidke muutujad avalikena. Nii on testimine lihtsam. Samuti tehke päisefail geometry.h, mille lisamisel lisatakse nii tipu, sirglõigu kui kera klassid.

Ülesanne 2 – standardteegi algoritmid ja funktsiooniobjektid

Üldised nõuded

Selle ülesande lahendamise käigus saate harjutada algoritmide ja funktsiooniobjektidega töötamist. Tulemusena tekkiv kood peaks sisalduma teegifailis nimega `libmyfunctors.a` ja vastav päis olgu `myfunctors.h`. Lahenduse testimiseks kirjutan ma programmi, kus rakendan teie kirjutatud koodi ning kontrollin tulemuste vastavust ülesande nõuetele. Soovitan teil teha testprogramm, kus oma lahendust ise järele proovite. Testprogrammi ei hinnata.

Lahendus paigutage samasse kataloogipuusse esimese ülesande geomeetriakoodiga. Makefile peab vaikimisi ehitama valmis mõlemad teegid. Võib juhtuda, et päris suur hulk koodi jääb päistesse ning vastavaid CPP faile ei olegi vaja. Proovige ise otsustada, kas päistest piisab või mitte.

Osa 1 – parameetriteta ja olekuta funktsiooniobjekt juhuarvude genereerimiseks

Kirjutada funktsiooniobjekt `RandomNumberGenerator`, mis tagastab juhuslikult genereeritud `unsigned long` tüüpi arvu. Funktsiooniobjekt peab sobima kasutamiseks algoritmide `generate` ja `generate_n` abil juhuslikke täisarve sisaldavate konteinerite (näiteks vektorite) loomiseks. Juhuslike arvude loomiseks kasutage funktsioone `rand` ja `srand` teegis `<cstdlib>`. Nende kasutamise kohta loe siit: http://www.cplusplus.com/reference/clibrary/cstdlib/. Tehke nii, et funktsiooniobjekt ei alustaks alati sama numbriga, samas vältige funktsiooni srand pidevat väljakutsumist.

Osa 2 – ühe parameetriga ja olekuta funktsiooniobjekt täisruutude äratundmiseks

Kirjutada ühe `unsigned long` tüüpi parameetriga predikaatfunktor `IsSquare`, mis tagastab tõese väärtuse, kui etteantud täisarv on mingi täisarvu ruut (näited sobivatest arvudest on 0, 1, 4, 9, 16,...). Funktor peab sobima kasutamiseks algoritmides `find_if`, `count_if`, `replace_if`, `remove_if`, `remove_copy_if`.

Osa 3 – ühe parameetriga olekuga funktsiooniobjekt konteineri elementide summeerimiseks

Kiire sisse juhatus olekuga funktorite teemasse. Funktorid võivad olla defineeritud ka enne nende rakendamist. Sellega kaasneb meeldiv lisavõimalus kasutada neid näiteks väärtuste kogumisel. Funktor luuakse enne algoritmi käivitamist ning pärast töö lõppu loetakse objekti seest tulemus. Selle toetamiseks peab funktori klassis olema lisaks operaatorile `()` defineeritud veel konstruktor ning vajalikud muutujad ning meetodid. Teie ülesandeks on kirjutada klassiparameetriga `T` klassimall `SumElements<T>`, mida saaks kasutada `T` tüüpi elemente sisaldavate konteinerite sisu

summeerimiseks. Eeldame, et tüübil `T` on defineeritud operaator `+` (sobivad näiteks `int`, `float`, `string`). Kui koodis proovitakse objekti luua mõne tüübiga, millel operaatorit pole, tekib viga.

Näide sellest, kuidas objekti peaks saama kasutada.

```
vector<unsigned long> values;             // vektor täisarvudest
SumElements<unsigned long> addThisValue;  // sama tüüpi funktor
addThisValue = for_each(values.begin(), values.end(), addThisValue); // summeeri
unsigned long sum = addThisValue.result ();          // loeme kogunenud summa
```

Kõigi ülesande osade lahendamine toob kokku kuni 3 punkti.

Lisaülesanne 1 – universaalne „hulknurk“

Tulemusena on vaja luua klassimall `Polygon`, mille parameetriteks on tipuvektori klass `T` ja täisarv `n` (`template <class T, int n>`). Klass esindab n-tipulisi hulknurki üle etteantud vektorite.

Lisage klassimallile meetodid:

Meetod	**Eesmärk**
`Polygon<T, n> ()`	algväärtustab tipud nullidega
`Polygon<T, n> (vector<T> pts)`	väärtustab vektori koordinaadid etteantud väärtustega
`float circumference ()`	arvutab hulknurga külgede pikkuste summa
`string toString ()`	tagastab hulknurka esitava sõne `((tipp1),...,(tipp2))`

Lahendus toob 1 lisapunkti.

Lisaülesanne 2 – väljundvoo operaatorid klassimallidele

Lisage universaalsele vektorile, sirglõigule, kerale ja hulknurgale väljundvoo operaator `<<`. Kasutage ära meetodit `toString`. Lahendus annab 1 lisapunkti.

Üldised tingimused

Tähtis! Loe läbi ülesannete vormistamise tingimused aine veebilehelt! Küsimustega pöörduda aine listi või praktikumijuhendaja poole. Tähtaeg on praktikumi toimumisnädala pühapäeva õhtu kell 23:59 (aega on 7 päeva).