LINKÖPINGS TEKNISKA HÖGSKOLA
Institutionen för datavetenskap
Avdelningen för programvara och system
Tommy Olsson

2014-05-26

Datortentamen i

TDDI14 Objektorienterad programmering

Provkod	DAT1
Datum	2014-05-28
Klockan	14-19
Plats	IDA, hus B, SU-salarna.
Jour	Tommy Olsson, 28 1954 Jourhavande besöker salarna varje hel timma under skrivtiden. *För övrigt ska jourhavande tillkallas endas i angelägna fall (t.ex. datorsystemproblem)!*
Administratör	Anna Grabska Eklund, 28 2362
Hjälpmedel	*En* bok om C++. Det får finnas egna marginalanteckningar i boken men ej på helt tomma sidor eller på lösa eller fastsatta extra blad eller lappar. Datorsystemets man-sidor är tillgängliga. Observera att det i enstaka fall kan finnas avvikelser från standarden – fråga jourhavande om problem tycks bero på att man-sida och kompilator verkar oense. En svensk-* ordbok får medtagas. Inga andra hjälpmedel, tryckta eller elektroniska, förutom kladdpapper och penna, är tillåtna.
Betygssättning	*Det finns tre uppgifter*. Totalt kan 20 poäng erhållas (10+5+5 poäng). För betyg 3 krävs 8 poäng, för betyg 4 krävs 12 poäng och för betyg 5 krävs 15 poäng.
Anvisningar	Se anvisningar på separat papper för inloggning till datortentamenssystemet. Logga inte ut och lås inte skärmen under tentamens gång. Logga ut först när du är klar med tentamen. Läs igenom hela uppgifterna noga innan du börjar lösa uppgifterna. Följ god kodningsstil så att dina program blir läsbara och begripliga. Skriv ditt namn och personnummer i en kommentar överst i varje fil. Skapa inga andra filer än de som efterfrågas i uppgifterna. Använd exakt de filnamn som anges. Skapa inga underkataloger för respektive uppgift, såvida det inte uttryckligen anges i en uppgift.

Lycka till!

1. Kopiera filen **uppgift1.cc** från given_files till din arbetskatalog. Se anvisningar som finns i filen, lägg till din egen kod.

Uppgiften är att konstruera en polymorf klasshierarki för att representera tvättmaskiner, torkmaskiner, diskmaskiner, etc. Följande klasser ska definieras:

- **Washing_Equipment** ska vara en gemensam abstrakt basklass. Alla maskiner har en *modellkod* (till exempel WE1486WM) och tillhör en *energiklass* (A+++, A++, A+, A, B eller C).
- **Washing_Machine** (tvättmaskin) ska vara en direkt konkret subklass till Washing_Equipment. Utöver modellkod och energiklass har tvättmaskiner även en *kapacitet* (kg, ett flyttalsvärde) och ett *varvtal* för den inbyggda centrifugen (rpm; ett heltalsvärde).
- **Clothes_Dryer** (torkmaskin) ska vara en direkt subklass till Washing_Equipment och gemensam abstrakt basklass för alla torkmaskinklasser. Inga specifika data är associerade med denna klass.
- **Condenser_Dryer** (kondenstorkmaskin) ska vara en konkret typ av torkmaskin. Utöver modellkod och energiklass har en kondenstorkare även en *kapacitet* (kg; ett flyttalsvärde).
- **Tumble_Dryer** (torktumlare) ska vara en annan konkret typ av torkmaskin. Inga specifika data är associerade med denna klass.
- **Dish_Washer** (diskmaskin) ska vara en direkt konkret subklass till Washing_Equipment. Utöver modellkod och energiklass har en diskmaskin även en *kapacitet*, räknat i antal kuvert som ryms (heltalsvärde), och en *ljudnivå* (dB; ett heltalsvärde).

All datamedlemmar ska initieras då ett objekt skapas och ska därefter aldrig ändras.

Utöver funktionalitet relaterad till att skapa och destruera objekt ska det finnas en **get**-*funktion* för varje datamedlem.

Objekt förutsätts alltid skapas dynamiskt (**new**) och hanteras via pekare.

Det ska *inte* finnas något annan publikt sätt att kopiera objekt än med en *polymorf funktion* **copy**(), som ska skapa en kopia av objektet ifråga och returnera en pekare till kopian. copy() ska använda kopieringskonstruktorn för att initiera kopian.

Tilldelning ska *inte* vara tillåtet alls.

Inga andra funktioner än de ovan nämnda får finnas!

Skriv ett testprogram enligt kommentarerna i den givna filen uppgift1.cc.

Följ specifikationen noga – följ anvisningarna i den givna filen noga – utnyttja C++ väl!

❑

2. Kopiera filen **uppgift2.cc** från given_files till din arbetskatalog. Följande definition av en **enum**-typ Step samt ett testprogram finns i filen.

```
enum Step { One, Two, Three };
```

Modifiera, på enklaste sätt, definitionen av Step så att One, Two och Three representeras av värdena 1, 2 respektive 3.

Överlagra **operator++** (prefix och postfix) för att stega upp Step-objekt från One till Two och från Two till Three. Om uppåtstegning utförs på ett Step-objekt som har värdet Three ska objektet inte ändras.

```
Step s{ One };
++s;                    // s stegas från One till Two
cout << s << endl;      // 2 skrivs ut (motsvarar Two)
s++;                    // s stegas från Two till Three
cout << s << endl;      // 3 skrivs ut (motsvarar Three)
++s;                    // s har värdet Three, stegas inte
cout << s << endl;      // 3 skrivs ut
```

Testprogrammet får alltså, för enkelhets skull, skriva ut den underliggande heltalsrepresentationen för uppräkningsvärdena. Fler tester finns i den givna filen.

Överlagra **operator--** (prefix och postfix) för att stega ned Step-objekt. Nedåtstegning av ett Step-objektet med värdet One ska inte ändra objektet.

❑

3. Kopiera filen **uppgift3.cc** från given_files till arbetskatalogen, skriv all kod i den filen.

Definiera en klassmall **Real** som är tänkt att *specialiseras* (*instansieras*) för en *flyttalstyp*, som **float**, **double** och **long double**, och ett *heltalsvärde* som anger precisionen för antalet decimaler, till exempel 8. Ett Real-objekt ska lagra ett värde av den flyttalstyp det har instansierats för och precisionen ska användas då Real-värden ska presenteras med hjälp av funktionen str(), se nedan.

När Real-objekt skapas ska de alltid *initieras* med ett värde (antalet decimaler i initialvärdet behöver dock inte överensstämma med den angivna precisionen och typen för initialvärdet kan vara en typ som är kompatibel med instanstypen).

```
Real<double, 8> r1{ 3.14159265 };      // lagrar double
```

Om ingen precision anges ska 6 vara *förvald precision*.

```
Real<float> r2{ 1.0 };                 // underförstått precision 6
```

Real ska ha en medlemsfunktion **str**(), som ska returnera en std::string med det lagrade värdet i strängform. Strängen ska genereras med fast decimalpunkt (fixed) och antalet decimaler ska överensstämma med precisionen. För objekten deklarerade ovan:

```
cout << r1.str() << endl;    // 3.14159265 skrivs ut (8 decimaler)
cout << r2.str() << endl;    // 1.000000 skrivs ut (6 decimaler)
```

str() ska definieras separat, dvs ska endast *deklareras* i klassdefinitionen och *definieras* utanför. Skulle du inte lösa det kan du definiera funktionen i klassen men får då avdrag med 1 poäng.

Real-objekt ska kunna initieras och tilldelas med objekt med identisk typ.

```
Real<double, 8> r3{ r1 };

Real<float> r4{ r2 };

r1 = r3;     // OK, samma typ.
r4 = r2;     // OK, samma typ.

r1 = r2;     // Fel, olika typer!

r1 = -1.0;   // Fel, olika typer!
```

Flyttsemantik (move-semantik) är inte intressant för denna typ av objekt och ska inte finnas.

Att **Real** endast ska kunna användas för flyttalstyper är inget som du ska lösa; du får anta att specialisering alltid görs med en lämplig typ och med ett meningsfull värde för precisionen.

❑