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

2014-08-18

Datortentamen i

TDDI14 Objektorienterad programmering

Provkod	DAT1
Datum	2014-08-21
Klockan	8-13
Plats	IDA, hus B, SU-salarna, plan 3.
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 anteckningar i boken men ej 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 två uppgifter. Totalt kan 20 poäng erhållas (10+ 10). 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 innan du börjar lösa.
	Följ god kodningsstil så att dina program blir läsbara och begripliga. Skriv 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, även med avseende på stora och små bokstäver.
	Skapa inga underkataloger.

Lycka till!

1. Kopiera filen **uppgift1.cc** från filkatalogen given_files till din arbetskatalog. Filen innehåller anvisningar för hur objekt ska hanteras och testas av programmet.

Konstruera en polymorf klasshierarki för att hantera kortläsare för digital-tv, som exempelvis CA-moduler (”conditional access modules”) och DTV-mottagare (digitaltvmottagare).

- Konstruera en abstrakt basklass **VC_Unit** (viewing card unit) som ska lagra korttillverkarens *namn* (string) och kortläsarens *pris* (**double**). När ett nytt VC_Unit-objekt initieras ska namnet på tillverkaren alltid ges. Priset är valfritt att ge vid initiering, om det inte ges ska det sättas till 0.0. Efter initiering ska endast priset kunna ändras.

  VC_Unit ska ha två direkta subklasser, CA_Module och DTV_Box.
- **CA_Module** ska vara en konkret klass för CA-modul-objekt och ska förutom tillverkarens namn och pris även lagra vilket *CA-system* (conditional access system; string) modulen är gjord för, till exempel Conax eller Viaccess. CA-system ska, tillsammans med övriga initialvärden, anges när ett CA_Module-objekt initieras och ska sedan inte ändras.
- **DTV_Box** ska vara en konkret klass för DTV-boxar, som förutom tillverkarens namn och pris ska lagra en *modellbeteckning* (string). Modellbeteckningen ska, tillsammans med övriga initialvärden, anges när ett DTV_Box-objekt initieras och ska sedan inte ändras.

  DTV_Box ska ha en subklass, DTV_Box_Recordable.
- **DTV_Box_Recordable** ska vara en klass för DTV-boxar med en inbyggd *hårddisk*, vilken används för att spela in program. Förutom tillverkarens namn, pris och modellbeteckning ska ett DTV_Box_Recordable-objekt även lagra *diskstorleken* (**int** som anger storleken i Gigabyte). Diskstorleken ska, tillsammans med övriga initialvärden, anges när ett DTV_Box_Recordable-objekt initieras och ska sedan inte ändras.

Det ska finnas en **get**-funktion för *varje* datamedlem i klasserna.

För *pris* enbart ska det finnas en **set**-funktion för att kunna ändra priset.

Objekt ska *inte* kunna *kopieras* eller *tilldelas*.

För *speciella medlemsfunktioner* gäller att, för varje klass, sådana som ska finnas och kan genereras av kompilatorn ska ”defaultas”, sådana som *inte* ska finnas men kan genereras av kompilator ska ”deletas”, sådana som *inte* ska finnas och *inte* genereras av kompilatorn ska inte deklareras.

Inga andra funktioner än ovan nämnda får finnas!

I den givna filen anges hur objekt ska deklareras/skapas och hur testning ska utföras.

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

❑

2. Kopiera filen **uppgift2.cc** från given_files till din arbetskatalog. Filen innehåller ett testprogram.

Konstruera en klassmall **Round_Robin**, som ska fungera som en cirkulär container för **operator**++.

- När ett nytt Round_Robin-objekt skapas ska elementtyp och maximal storlek anges som mall-argument.

```
Round_Robin<int, 10> rr;      // int-element, maximal storlek 10, initial tom
```

Tips: Konstruera Round_Robin först som icke-mall, med fix elementtyp och storlek, mallifiera sedan.

- std::vector ska användas för att lagra element internt.
- En intern *aktuell position* ska ange ett av elementen i vectorn. Använd vector::iterator för det.

Tips: Deklarationen av vector-iteratorn kommer att vara beroende av mallparametern för elementtypen; **typename** behöver användas i deklarationen.

- funktionen **size**() ska returnera den aktuella storleken, dvs antalet lagrade element.
- funktionen **max_size**() ska returnera den maximala storleken Round_Robin-objektet kan ha.

```
cout << rr.size() << endl;
cout << rr.max_size() << endl;
```

- funktionen **empty**() ska returnera **true** om Round_Robin-objektet är tomt, annars **false**.
- funktionen **add**(x) ska lägga till värdet x genom att sätta in det *sist* i vectorn och sedan sätta den aktuella positionen till det *första* elementet i vectorn. Om Round_Robin-objektet är fullt ska i stället ett egendefinierat undantag **round_robin_error** kastas med meddelandet ”Round_Robin: full”.

```
rr.add(4711);
```

Definitionen för add() ska ges separat från definitionen för Round_Robin.

- funktionen **reset**() ska återställa den aktuella positionen till det första elementet i vectorn, om sådant finns, annars något annat lämpligt värde.

```
rr.reset();
```

- **operator*** ska returnera en referens till det aktuella elementet; om Round_Robin-objektet är tomt får man skylla sig själv om man försöker använda **operator*** (odefinierat beteende).

```
cout << *rr << endl;     // värdet på aktuellt element skrivs ut
*rr = 11147;             // värdet på aktuellt element ändras till 11147
```

- **operator**++ ska stega fram den aktuella positionen ett element, *cirkulärt* (dvs om den aktuella positionen anger det sista elementet i vectorn ska den nya aktuella positionen ange det första elementet). Både prefix- och postfixform ska definieras med gängse semantik.

Definitionerna för **operator**++ *ska ges separat från definitionen för Round_Robin*

- *Defaultkonstruktorn* ska initiera ett Round_Robin-objekt till tom container.
- Typlika Round_Robin-objekt ska kunna kopieras och tilldelas.
- Speciella medlemsfunktioner behöver bara deklareras ifall de kompilatorgenererade versionerna *inte* fungerar som de bör för Round_Robin-objekt.

round_robin_error ska härledas från någon lämplig standardundantagsklass i exceptionhierarkin, med tanke på felets karaktär och den funktionalitet som ska finnas för hantering av ett felmeddelande.

Vänd!

Utskriften från programmet ska vara följande:

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
1 2 3 4 5 6
1 2 3 4 5 6 7
1 2 3 4 5 6 7 8
1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9 10
Round_Robin: full
1 3 5 7 9 1 3 5 7 9 1
```

❑