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

2014-10-09

Datortentamen i

TDDI14 Objektorienterad programmering

Provkod	DAT1
Datum	2014-10-21
Klockan	8-13
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 marginalen men ej på helt tomma sidor, 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, på papper eller elektroniska, förutom kladdpapper och penna, är tillåtna.
Betygssättning	Totalt kan 20 poäng erhållas (8+8+4). 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 uppgiften innan du börjar lösa den. 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, även med avseende på stora och små bokstäver. Skapa inga underkataloger.

Lycka till!

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

Definiera en polymorf klasshierarki för att hantera olika modeller av grillar. Grillobjekt ska alltid skapas dynamiskt (med **new**) och alltid hanteras via pekare.

- Klassen **Grill** ska vara en abstrakt basklass för alla grillklasser.

  Alla grillmodeller ska ha ett *modellbeteckning*, till exempel LPG-4710, som ska lagras som std::string. Modellbeteckningen ska alltid anges när ett nytt grillobjekt skapas och ska sedan aldrig ändras.

Det finns två huvudkategorier av grillar, *kolgrillar* och *gasolgrillar*, och det ska finnas två motsvarande konkreta klasser, Grill_Charcoal och Grill_LPG (”Liquid Petroleum Gas”). Dessa ska vara direkta subklasser till Grill.

- För **Grill_Charcoal** ska ingen annan information än modellbeteckningen lagras. Modellbeteckningen ska alltid anges i samband med att ett nytt Grill_Charcoal-objekt skapas.
- För **Grill_LPG** ska antalet *brännare* och den *gastyp* som ska använda lagras.

  Antalet brännare kan variera mellan 2–6, lagras som **int**. Information om den *gastyp* som ska användas, till exempel propan eller butan, ska lagras som std::string.

  Modellbeteckning och antalet brännare ska alltid anges när ett nytt Grill_LPG-objekt skapas och ska sedan inte ändras. Om man inte anger gastyp när ett nytt objekt skapas ska defaultvärdet vara ”Propan”.
- Vissa gasolgrillar finns i en variant med *sidobrännare*. Sådana grillar ska representeras av klassen **Grill_LPG_Sideburner**, vilken ska vara en direkt subklass till Grill_LPG.

  Inga data tillkommer jämfört med Grill_LPG. Sidobrännaren räknas *inte* in i det antal brännare som ska anges för Grill_LPG_Sideburner. Samma sätt att initiera nya objekt av typen Grill_LPG_Sideburner ska gälla som för objekt av typen Grill_LPG, se ovan.

För varje datamedlem (modell, antal brännare, gastyp) ska det finnas en motsvarande **get**-funktion som returnerar värdet.

Grill-objekt ska *inte* kunna *kopieras* eller *tilldelas*, i någon form.

För *speciella medlemsfunktioner* gäller, för *varje* klass, att 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 göras.

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

❑

2. Kopiera filen **uppgift2.cc** från filkatalogen given_files till arbetskatalogen. Filen innehåller en rudimentär definition av en klass Adder, se nedan, och ett till största delen bortkommenterat testprogram.

```
class Adder
{
public:
  Adder(int i = 0) : value_{ i } {}
private:
  int value_;
};
```

Komplettera klassen Adder så att följande funktionalitet uppnås:

- ett heltalsvärde ska kunna adderas till ett Adder-objekt med **operator**+=

  `a1 += 1` `// datamedlemmen a1.value_ ska ökas med 1`

- ett Adder-objekt ska kunna adderas till ett annat Adder-objekt med **operator**+=

  `a1 += a2` `// a1.value_ ska ökas med värdet av a2.value_`

- två Adder-objekt ska kunna adderas med **operator**+

  `a + b` `// ska vara ett Adder-objekt vars värde är a1.value_ + a2.value_`

- ett Adder-objekt ska kunna tilldelas ett heltalsvärde med **operator**=

  `a1 = 2` `// a.value_ sätts till 2`

- ett Adder-objekt ska kunna tilldelas ett annat Adder-objekt med **operator**=

  `a1 = a2` `// a1.value_ sätts till a2.value_`

- man ska kunna skriva ut det lagrade värdet (value_) hos ett Adder-objekt med **operator**<<

  ```
  cout << a1 << endl;
  ```

- Ett nytt Adder-objekt ska kunna initieras med ett heltalsvärde eller med ett annat befintligt Adder-objekt.

När all funktionalitet fungerar för Adder och **int**, gör om Adder till en klassmall så att datatypen för value_ kan väljas med en malltypparameter och anpassa all kod till detta.

Samtliga *speciella medlemsfunktioner ska finnas.* ”Defaulta” eller definiera själv, beroende på vad som krävs.

❑

3. Kopiera filen **uppgift3.cc** från filkatalogen given_files till arbetskatalogen. Filen innehåller ett testprogram.

I standardbiblioteket (inkludering <utility>) finns std::pair, som är en klassmall för att hantera värdepar. Man kan till exempel deklarera ett objekt av typen pair för ett ett **char**-värde och ett **int**-värde.

```
pair<char, int> p('A', 1);
```

Klassen pair har två publika datamedlemmar, *first* och *second*, i vilka de två värdena lagras.

- Definiera en funktionsmall **print_pair**, som kan skriva ut ett pair på formen *first*`;`*second* på en utström, dvs de två värdena separerade med ett *semikolon*. Funktionen ska ta en utström och ett pair som argument.

  Du får förutsätta att **operator**<< kan användas för att skriva ut värdena för first och second.

Korrekt löst **print_pair** ger **1 poäng**.

- Definiera en funktionsmall **read_pair**, som kan läsa in ett pair från en inström, under förutsättning att de två värdena separeras med ett *semikolon*. Funktionen ska ta en inström och en variabel av typen pair som argument.

  Felkontroller på inmatade värden behöver ej göras – det är tillåtet att anta att enbart korrekta värden matas in.

  Du får förutsätta att **operator**>> kan användas för att läsa in värdena för first och second.

För 2 poäng på hela uppgiften behöver **read_pair** enbart fungera då första värdet i ett inmatat par är av grundläggande typ, till exempel **char**, **int**, **double** eller **bool**, inte då det är ett strängvärde (std::string eller **char***).

För 4 poäng på hela uppgiften måste **read_pair** fungera även då första värdet i ett par är ett strängvärde. Vita tecken förutsätts *inte* förekomma i strängvärden. Semikolon förutsätts *inte* heller förekommer i strängvärden, utan kan alltid tolkas som en avslutare/separator vid läsning av par där det första värdet är ett strängvärde.

❑