TDIU20 – Tentaregler

Inloggning

Logga in i tentasystemet genom att välja session ”exam system” och logga in med ditt vanliga LiU-ID. Välj inte att ha denna session som standardsession. Verifiera att dina uppgifter stämmer och förbered din tentaplats. Som vanligt är det inte tillåtet att ha väskor eller jackor vid sin skrivplats och mobiltelefoner ska ligga avstängda i jacka eller väska. Ta fram ditt LiU-kort och invänta tentavakt för att få ett engångslösenord.

Hjälpmedel

Följande får tas med på tentan:

- En bok om c++. För boken gäller följande regler:
  - Kommentarer/noteringar som direkt rör text och exempel på sidan i fråga får finnas i sidmarginalen.
  - Egna sidflikar för att enkelt kunna hitta t.ex. de olika kapitlen är tillåtna.
  - Inga extra ark eller lappar, lösa eller fastsatta, får finnas.
  - Tomma sidor, insidan av pärmarna, försättsblad, etc., får inte innehålla programkod.
- Ett A4-ark med valfritt innehåll på vardera sidor (går ej att ersätta med två enkelsidiga ark).
- Penna för att anteckna under tentan. Ni kommer förses med blanka papper

Följande får INTE tas med:

- Elektroniska hjälpmedel såsom miniräknare och mobiltelefon

Utloggning

När du är nöjd med ditt betyg (som står i tentaklienten) kan du avsluta och logga ut som vanligt i menyn. Klicka sedan på ok följt av knappen avsluta tentan. Observera att du när du gjort detta inte kan logga in igen. Lämna inte din plats innan vanliga inloggningsskärmen syns.

Tentaregler

Tentan består av fyra uppgifter klassificierade som grundnivå eller avancerad. För godkänt betyg på tentan krävs lösning av en uppgift på grundnivå. För betyg 4 krävs dessutom lösning av en avancerad uppgift och för betyg 5 krävs en grunduppgift samt båda avancerade uppgifterna.
För att en uppgift ska anses godkänd krävs följande:

- att man noga följt alla instruktioner och krav ställda i uppgiften
- din kod följer god programmeringsstil (se labseriens rättningsguide)
- att klasser har ett tydligt ansvar och funktioner har en väl definierad uppgift
- att din kod har bra inkapsling och resurshantering

Bonus från labserien

Om du har bonus från labserien minskas antalet avancerade uppgifter som krävs med ett, dvs för betyg 4 krävs endast en grunduppgift och för betyg 5 krävs en av varje. Bonusen är endast giltig det år den erhölls.

Frågor om uppgifter

Frågor om tentan i stort eller uppgiftspecifika frågor ska ställas via tenta-klienten. Detta för att vi ska ha en historik av konversationen samt för att vi ska kunna ge samma hjälp till olika studenter.

Systemfrågor

Om du har systemfrågor som t.ex. problem med tentaklienten eller terminalen räcker du upp handen så kommer en assistent och hjälper till.

Uppgift 1 - Grundnivå

Skriv din lösning på en fil med namnet `uppgift1.cc`

I denna uppgift ska du skapa klasser som beskriver ett kvitto. Ett kvitto har ett kvittonummer och består av ett antal produktrader. En produktrad består i sin tur av, produktbeteckning, antal och styckpris.
Skapa en klass `Product_Line` för att representera en produktrad. Skapa därefter en klass `Receipt` som innehåller en `vector` av `Product_Line`. Kvittoklassen ska ha en medlemsfunktion `add_product` för att registrera en ny produkt på kvittot, en `print_receipt` för att skriva ut kvittot samt en funktion `purchase_value` för att beräkna kvittots värde (summan av alla inköpta produkter).
Ditt program ska skapa kvitton och registrera de produktrader som behövs för att sedan kunna skriva ut kvitton enligt körexemplet nedan. Du ska alltså hårdkoda in produktraderna och sedan skriva ut de tre kvittona.

Körexempel 1

```
Receipt no 1:
apelsin: 2st * 5.00kr
banan: 5st * 7.50kr
citron: 1st * 15.00kr
===
Köpeskilling: 62.50kr

Receipt no 2:
gurka: 1st * 9.00kr
tomat: 5st * 18.50kr
sallad: 1st * 11.00kr
===
Köpeskilling: 112.50kr

Receipt no 3:
smör: 1st * 19.00kr
bröd: 10st * 24.50kr
ost: 1st * 48.00kr
===
Köpeskilling: 312.00kr
```

OBS: Uppgiften går ut på att skapa klasserna enligt beskrivningen. Inte att endast få samma utskrift som körexemplet. Försäkra dig om att all kod du skriver används (testas) på ett bra sätt innan du lämnar in.

Uppgift 2 - Grundnivå

Skriv din kod på en fil med namnet `uppgift2.cc`

Skapa en klass som representerar en produkt. En produkt består av ett namn, förpackningens vikt (kg), och förpackningens pris (kr).

En produkt ska vara möjlig att jämföra mot en annan produkt med den vanliga operatorn <, och gå att läsa in med vanlig operator för formaterad inläsning `>>` samt skriva ut med `<<`. Jämförelsen sker på produktens kilopris.

Huvudprogrammet ska läsa in tre produkter och därpå skriva ut dem i mest prisvärda ordning, mest prisvärd (lägst kilopris) först. I andra hand används produktnamnet (bokstavsordning). Sorteringen kan exempelvis lösas med tre jämförelser.

En produkt matas in och skrivs ut enligt givna körexempel.

Körexempel 1 (Användarinmatning är i fet stil)

```
Mata in produkt 1: Potatis; 5 kg; 49 kr;
Mata in produkt 2:     Morötter; 0.5 kg; 13 kr;
Mata in produkt 3: Lök; 0.1 kg; 2 kr;

Produkterna ordnade med mest prisvärd produkt först:
Potatis, 9.80 kr/kg
Lök, 20.00 kr/kg
Morötter, 26.00 kr/kg
```

Körexempel 2 (Användarinmatning är i fet stil)

```
Mata in produkt 1: Citron; 0.2 kg; 10.0 kr;
Mata in produkt 2: Bananer; 2 kg; 49.0 kr;
Mata in produkt 3: Apelsiner; 1 kg; 24.5 kr;

Produkterna ordnade med mest prisvärd produkt först:
Apelsiner, 24.50 kr/kg
Bananer, 24.50 kr/kg
Citron, 50.00 kr/kg
```

TIPS: Använd getline med semikolon som radslutavgränsare för att läsa in produkterna. Se upp så du inte får med blanksteg eller nyradstecken före produktnamnet.

Uppgift 3 - Avancerad

Kopiera filen `given_files/uppgift3.cc` till din hemmapp och lägg till din lösning i denna. Observera att du inte får göra ändringar i de givna funktionerna.

Ibland behöver man göra en transaktion utan att själv behålla en kopia. Lämnar jag betalning för t.ex. en get är det förstås naturligt att betalningen övergår helt till ägaren, utan att jag senare kan påverka den.
Skapa en klass för en sedel, med namn `Banknote`, som ser till att värdet av ett objekt förs över helt vid alla typer av kopiering och flytt så sedeln inte kan användas igen. När ett `Banknote`-objekt nyskapas (inte kopieras eller flyttas) ska värdet skrivas ut, och när ett objekt försvinner ska värdet skrivas ut igen, men *bara* om objektet inte använts (flyttats eller kopierats).

Se det givna programmet för exempelutskrift och förklaring.

TIPS: Tänk noga igenom vilka medlemsfunktioner som anropas i de olika stegen i den givna koden för att reda ut hur denna klass ska fungera.

Uppgift 4 - Avancerad

Kopiera filen `given_files/uppgift4.cc` till din hemkatalog och modifiera denna. Filen innehåller ett givet huvudprogram som inte får ändras.

I denna uppgift ska du skapa en polymorfisk klasshierarki för att representera olika betalmedel. Alla betalmedel har en utfärdare och kan ibland vara elektroniska. Det finns två kategorier av betalmedel, de som är kopplade till ett konto hos utfärdaren (kort), och de som distribueras (mynt, sedlar) där varje enhet garanteras ett nominellt värde av utfärdaren.

Designa din klasshierarki enligt följande beskrivning:

- Skapa en basklass med namn `Payment_Method` med datamedlemmar `std::string issuer` och `bool electronic`. Det ska inte gå att skapa objekt av denna typ.
- Skapa en direkt subklass till `Payment_Method` med namn `Distributed_PM` för att representera distribuerade betalmedel. När en `Distributed_PM` skapas ska den ta emot information om utfärdaren samt om den är elektronisk. Dessutom ska den lagra betalmedlets nominella värde (dess värde vid betalning). Även detta ska kunna anges när en `Distributed_PM` skapas. Om inget anges är betalmedlet *inte* elektroniskt. Det ska inte gå att skapa objekt av denna typ.
- Skapa en subklass till `Distributed_PM` med namn `Coin`. Ett mynt har både ett nominellt värde (värdet vid betalning) och faktiskt värde (metallvärdet). Ett mynt är aldrig elektroniskt.
- Skapa en annan direkt subklass till `Payment_Method` med namn `Account_PM` som alltid är elektroniskt. Förutom utfärdare ska `Account_PM` initieras med en ägare (givet som en `string`). Det ska inte gå att skapa objekt av denna typ.
- Skapa en subklass till `Account_PM` med namn `Card`. Ett kort har förutom utfärdaren och ägaren ett kortnummer.
- Kopiering och flytt av objekt i denna hierarki ska inte vara tillåtet på något sätt.

Förutom de speciella medlemsfunktioner som krävs ska samtliga klasser i hierarkin ha följande medlemsfunktioner:

- `string name()` ska returnera namnet (hårdkodat) på klasstypen.
- `void print()` ska skriva ut en beskrivning av objektet. Först ska klasstypens namn skrivas ut, och sedan informationen från `Payment_Method`. Därefter skrivs olika saker ut beroende på typ av objekt enligt nedan:
  - För `Coin`: Skriver ut även nominellt och faktiskt värde.
  - För `Card`: Skriver ut även ägare och kortnummer.

OBS: Uppgiften går ut på att skapa klasserna enligt beskrivningen. Inte att endast få samma utskrift som körexemplet.
TIPS: Tänk igenom hur public/private/protected fungerar och vad som krävs för att huvudprogrammet (inte) ska kunna skapa objekt av klassen.

Körexempel 1

```
Card (electronic): VISA; owner = Kim; number = '4711 0123 9876 0001';
Coin (real money): Riksbanken; nominal value = 10;  real value = 0.12;
```