TDP004 - Tenta

2021-01-14

Regler

- All kod som skickas in för rättning ska kompilera och vara väl testad.
- Du ska sitta i ett rum själv.
- Du ska ha komeran igång under hela tentamenstiden
- Kameran ska visa dig och din omgivning tydligt.
- Vid toalettbesök ska detta meddelas genom att sätta fram en lapp framför kameran så att tentavakten kan se både lappen och rummet.
- All form av kontakt mellan studenter under tentamens gång är strängt förbjuden.
- Det är inte tillåtet att söka information via externa webbplatser, se tabellen nedan för tillåtna hjälpmedel.
- Frågor om specifika uppgifter eller om tentamen i stort ska ställas via tentasystemets kommunikationsklient.
- Systemfrågor kan ställas till assistent genom att räcka upp handen i Zoom.
- Endast uppgifter inskickade före tentamenstidens slut rättas.
- Ingen uppgift kan kompletteras under tentamens sista kvart.
- En uppgift kan som regel kompletteras tills den är antingen "Godkänd" (alltså fått 2 poäng) eller "Underkänd". En uppgift bedöms som "Underkänd" om ingen markant förbättring skett sedan tidigare inlämning.
- Kompilerande kod, fullständig kravuppfyllnad och följande av god stil och goda konventioner enligt god programmeringssed är krav för att en uppgift ska bedömas "Godkänd".

Hjälpmedel	En C++-bok (t.ex. Stroustrup) Ett A4-ark med egna anteckningar en.cppreference.com

Uppgift 1 - Klasser - Lagerlokaler

Federationen har många saker, dessa saker behöver lagras någonstans. Janeway har därför fått i uppdrag att skapa ett mindre system för att hantera lager. Som vanligt faller det till dig att lösa de faktiska problemen. Kortfattat kan vi säga att `Saker` i denna uppgift representeras av ett namn och ett värde. Exempelvis kan en sak ha namnet `pennor` och värdet `15000`. `Lager` lagrar ett godtyckligt antal `Saker`.

I den givna filen `uppgift1.cc` finns ett antal kommentarer som beskriver vad ditt huvudprogram ska göra. Du behöver också skriva klassen eller klasserna som krävs för att lösa problemet.

Följande krav skall uppfyllas (utöver de vanliga krav vi ställer på era lösningar och design av klasser):

1. Ett lager ska representeras av en klass
2. Sakerna som lagras i lagren skall representeras av **en** lämplig datatyp (användardefinerad eller inte)
3. Utskrift av ett lager ska ske med ett anrop till en medlemsfunktion från huvudprogrammet
4. Sammanslagning av två lager ska ske med ett anrop till en medlemsfunktion från huvudprogrammet
5. Utskriften och sammanslagningen av lagerlokaler skall gå att göra med överlagrade operatorer (bara för 2p)

Körexempel:

```
Warehouse 1:
Boardgames: 7250
Locks: 7500

Warehouse 2:
Cars: 100000
Pens: 15000
Boardgames: 2500

Warehouse combined:
Boardgames: 9750
Locks: 7500
Cars: 100000
Pens: 15000
```

Uppgift 2 - Polymorfi - Quiz frågor

Picard vill skapa ett quiz program som kan generera godtyckliga typer av frågor. Till en början känner Picard att det räcker med att ha frågor där användaren får svara med fritext eller välja ett svarsalternativ. Detta skulle dock kunna ändras i framtiden med flera typer av frågor. Han vill därför att du skapar en arvsstruktur för frågor. Klasserna för strukturen är följande:

`Question` (basklass):

- En `Question` har alltid en titel och en beskrivning (frågan).
- `Question` ska alltid kunna skriva ut sin titel samt sin beskrivning på ett snyggt sätt, se körexempel.
- Det ska finnas en metod för hantering av frågan och svaret. Hanteringen har inget standardbeteende.

`Textual_Question` (subklass):

- `Textual_Question` ska utöver titel och beskrivning spara svaret till frågan.
- Hanteringen sker genom att:
  1. Läsa in svaret från användaren
  2. Returnera sant eller falskt beroende på om användaren svarade rätt eller fel
- Även `Textual_Question` ska skriva ut sin typ (“Textual question”) i början av utskriften, se körexemplet nedan.

`Multi_Question` (subklass):

- `Multi_Question` har alltid utöver titel och beskrivning även sparat valen till frågan samt det rätta svaret
- Hanteringen av en `Multi_Question` sker genom:
  1. Skriva ut alternativen
  2. Läsa in det alternativ som användaren valt, användaren bör svara med indexet av alternativet
  3. Returnera sant eller falskt beroende på om användaren svarade rätt respektive fel
- `Multi_Question` ska även början utskriften av frågan med vilken typ den är, dvs ”Multi question”, se körexemplet nedan.

Utöver klasserna ovan behöver även du implementera ett huvudprogram. Det finns i nuläget en beskrivning av huvudprogrammet i filen `uppgift2.cc`. Tänk på att huvudprogrammet ska fungera för fler än bara två frågor genom att endast lägga till fler frågor i behållaren.
Tips för hantering av frågor, ni kan använda er av de inbyggda jämförelse operatorerna för att se om svaret är rätt eller inte. Exempelvis så kan ni använda strängjämförelse i `Textual_Qeustion`.

Följande krav skall uppfyllas (utöver de vanliga krav vi ställer på era lösningar):
Krav: Programmet ska inte ha minnesläckor.
Krav: Frågorna ska kunna skriva till och läsa från godtyckliga strömmar (bara för 2p)

Körexempel:

```
Textual question: Question 1

Enter your answer: TDP004
Your answer was: Correct!

Multi question: Question 2

Which one of these datatypes represents a floating point number?

Select the correct alternative:
1. int
2. char
3. double
4. string

Enter alternative index: 3
Your answer was: Correct!
```