TDP004 - Dugga

2019-12-06

Regler

- All kod som skickas in för rättning ska kompilera och vara väl testad.
- Inga elektroniska hjälpmedel får medtas. Mobiltelefon ska vara avstängd och ligga i jacka eller väska.
- Inga ytterkläder eller väskor vid skrivplatsen.
- Student får lämna salen tidigast en timme efter tentamens start.
- Vid toalettbesök ska pauslista utanför salen fyllas i.
- All form av kontakt mellan studenter under tentamens gång är strängt förbjuden.
- Böcker och anteckningssidor kan komma att granskas av assistent, vakt eller examinator under tentamens gång.
- Frågor om specifika uppgifter eller om tentamen i stort ska ställas via tentasystemets kommunikationsklient.
- Systemfrågor kan ställas till assistent i sal genom att räcka upp handen.
- 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" 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

Information

Betygssättning vid tentamen

Tentamen består av ett antal uppgifter på varierande nivå. Uppgifter som uppfyller specifikationen samt följer god sed och konventioner ges omdömet "Godkänd". Annars ges omdömet "Kompletteras" eller "Underkänd". Tentamen kräver två godkända uppgifter för betyg 3. Alla betygsgränser ses i tabell 1 och 2. *För betyg 3 har du alltid hela tentamenstiden, varken mer eller mindre.* (För student som från LiU fått rätt till förlängd skrivtid förlängs betygsgränserna i proportion till den förlängda skrivtiden.)

Tid	**Lösta uppgifter**	**Betyg**	**Tillgodo**
4 h	3	5	inget
2.5 h	2	4	inget
4 h	2	3	inget
4 timmar	1	|	löst uppgift

Tabell 1: Betygsättning vid förtentamen (dugga), 4 uppgifter ges

Tid	**Lösta uppgifter**	**Betyg**
3 h $+B$	3	5
4 h $+B$	4	5
4 h $+B$	3	4
2 h $+B$	2	4
5 timmar	2	3

Tabell 2: Betygsättning vid sluttentamen, 5 uppgifter ges

Bonustid ($+B$)

All bonustid gäller endast under den första ordinarie tentamen i samband med kursen (januari). Varje moment i kursen som ger bonus ger 5 minuter extra tid för högre betyg på sluttentamen, upp till maximalt 45 minuter. Detta är markerat med $+B$ i tabellen där bonus räknas.

Tillgodoräknanden

Tillgodoräknanden gäller endast under den första ordinarie tentamen i samband med kursen. Om du på förtentamen endast lyckas lösa en uppgift kan du tillgodoräkna motsvarande uppgift på sluttentamen (se tabell 1). Du har då bara en uppgift kvar till betyg 3. För högre betyg (om du löser mer än en uppgift på förtentamen) kan du inte tillgodoräkna något. Om du på sluttentamen löser en andra uppgift snabbt och vill sikta på högre betyg kan du lösa den tillgodoräknade uppgiften "igen", men den räknas endast mot högre betyg, **inte** som andrauppgift för betyg 3.

Inloggning

Logga in på datorn i labbsalen med ditt liu-id och lösenord. Detta är samma inloggningsuppgifter som du använder i Lisam.

Skrivbordsmiljön

När du kommit in i tentasystemet har du en normal skrivbordsmiljö (Mate-session i Ubuntu). Efter en stund kommer även din kommunikationsklient att dyka upp automatiskt. Startmenyn är nedskalad till enbart det som examinator bedömt relevant. Andra program kan fortfarande finnas tillgängliga genom att starta dem från en terminal. Observera att en del program som använder nätverkstjänster inte fungerar normalt, eftersom nätverket inte är åtkomligt.

När du är inloggad är det viktigt att du har tentaklienten igång hela tiden. Om den inte dykt upp fem minuter efter inloggning och inte heller när du startar den manuellt från menyn (fisken) tar du kontakt med assistent eller vakt i sal.

Terminalkommandon

`e++17` används för att kompilera med "alla" varningar *som fel*.
`w++17` används för att kompilera med "alla" varningar. **Rekommenderas.**
`g++17` används för att kompilera `utan` varningar.
`valgrind --tool=memcheck` används för att leta minnesläckor.

C++ referenssidor

På tentan har du *experimentiell* tillgång till referenssidorna på `http://www.cppreference.com/` via webbläsaren Chrome. Starta `chromium-browser` i terminalen eller välj lämpligt alternativ från startmenyn. Observera att allt utom referenssidorna är avstängt. Om du inte kan komma åt en sida du tycker hör till referenssidorna (som kanske blockerats av misstag) kan du skicka ett meddelande via tentaklienten. Då lösningen är på experimentell nivå kan det hända att problem uppstår, t.ex. att proxyn inte går att nå. Tag då hjälp av assistent i sal.

Givna filer

Eventuella givna filer finns i katalogen `given_files`. Denna underkatalog är skrivskyddad, så det är ingen risk du råkar ändra på dessa filer. Skrivskyddet gör dock att du måste kopiera in de givna filer du vill använda till tentakontots hemkatalog. Hur du listar och kopierar filer ska du kunna. Hemkatalogen står du i från början, och du kommer alltid tillbaka till den genom att bara exekvera kommandot `cd` i terminalen.

Avslutning

När dina uppgiftsbetyg och ditt slutbetyg i kommunikationsklienten stämmer överens med det du förväntar och du är nöjd, eller när tentamenstiden är slut, är det dags att logga ut. Hinner du inte se ditt betyg får du höra av dig till examinator via epost efter tentamen.

Avsluta alla öppna program och logga ut ur datorn. Lämna inte datorn innan du ser att du är utloggad.

Uppgift 1 - Representera en samling soldater

Kapten Janeway försöker komma fram till hur stark hennes samling av miniatyrsoldater är i spelet Frostgrave. Hon kan programmera lite och har skrivit ett C++ program som hon tycker är bra för att lösa detta problem. Tyvärr så beror detta program på att vissa klasser existerar som inte finns i standardbiblioteket. Läs hennes huvudprogram och skapa klasserna som behövs för att det skall köra enligt körexemplet.

`Soldier` har datamedlemmarna `name` (namnet som en text) och `power` (kraften som ett heltal). `Warband` har datamedlemmarna `name` (namnet som en text) och `soldiers` (alla soldater som ingår i objektet, sparade i en lämplig behållare). För att jämföra två `Warband` så summeras kraften hos alla soldaterna i `Warband`, det objekt som har störst total kraft anses vara kraftfullast. Observera att det finns två sätt att lägga till soldater i `Warband`, antingen genom att anropa `add_soldier` med en `Soldier` som parameter eller med en `string` och `int` som parametrar.

Du ska lämna in **3** filer. `warband.h`, `warband.cc` och `uppgift1.cc`. Du skall deklarera och implementera dina klasser i de två förstnämnda filerna enligt god programmeringssed.

Krav:

- Det skall gå att lagra ett godtyckligt antal `Soldier` i `Warband`.
- Du får inte lagra onödiga datamedlemmar
- Du får inte duplicera kod eller bryta inkapsling i onödan.
- Du får inte ändra i den givna koden.

Körexempel:

```
$ ./a.out
Warband 2 is the strongest warband
Warband 1 is the strongest warband
```

Uppgift 2 - Cisco har vill ha ett speciellt magisystem

Kapten Cisco håller på att utveckla regler för hur trollkonster ska fungera i spelet Frostgrave. Han vill att trollkonster ska fungera lite annorlunda i sin spelvärld, de ska nämligen bara kunna finnas på ett ställe samtidigt. En trollkonst kan alltså finnas nedskriven i en formelbok **eller** vara inlärd av en person **eller** flyta runt i etern.

Cisco har implementerat ett par klasser som representerar trollkonsten. Varje trollkonst (`Spell`) består av ett antal effekter (`Effects`). Pekare till effekterna lagras i en `std::stack`. Problemet är att trollkonster nu äger pekare och inte har tagit ansvar för den minneshanteringen genom att överlagra de speciella medlemsfunktionerna, detta är ditt jobb.

Du ska se till att de speciella medlemsfunktionerna fungerar korrekt. Trollkonster ska gå att flytta, men inte kopiera. Med andra ord är det ok att flytta en trollkonst från en variabel till en annan men inte göra en kopia av en trollkonst. Du ska också se till att huvudprogrammet inte läcker minne när du är klar. Det ska också gå att anropa medlemsfunktionen `cast` på en magi och skicka med valfri `std::ostream`. `cast` ska skriva varje `Effect` i stacken till strömmen enligt körexemplet och ta bort dessa ur stacken.

Krav:

- Programmet ska köra enligt körexemplet
- Du får inte ändra i den givna koden
- Du ska explicit specificera beteendet för de speciella medlemsfunktionerna
- Medlemsfunktionerna skall fungera som förväntat även om beteendet inte testas

Tips:

- `std::stack` har ett par användbara medlemsfunktioner som du kan behöva:
  - top() -> returnerar elementet som ligger på toppen av stacken
  - pop() -> tar bort elementet högst upp på stacken, inget returvärde

Körexempel:

```
$ ./a.out
Correct, the spell can not be copied using the assignment operator(=)
Correct, the spell can not be copied using the copy constructor

Casting Voldos awesome incantation: Teleport -> Levitate
```

Uppgift 3 - Hjälp Archer sparka rätt undersåtar

Kapten Archer spelar för närvarande figurspelet Frostgrave. I detta figuspel spelar han den ondskefulle trollkarlen Ulagast. Ulagast leder ett antal undersåtar (representerade av klassen `Minion`). Tyvärr har Archer samlat på sig lite för många undersåtar och behöver bestämma sig för vilka han skall göra sig av med.

Undersåtarna som Archer har finns representerade i `vector<Minion> minions` i huvudprogrammet. Ditt jobb är att skriva ett program som med hjälp av standardbiblioteket i C++ skapar två nya vectorer med `Minions`. Den första vectorn skall heta `minions_to_keep` och den andra ska heta `minions_to_fire`.

De minions som är värdefulla nog skall sparas i `minions_to_keep` och de som inte är det ska läggas i `minions_to_fire`. Hur vectorn `minions` ser ut efter körning spelar ingen roll. En minions värde beräknas av deras `power` multiplicerat med deras `evilness`. En minion är värd att behålla om värdet är större än 14. Innehållet i dessa behållare skall sedan skrivas ut i enlighet med körexemplet.

Krav:

- Du får ändra på klassen Minion men den skall fungera med den givna koden. Om klassen inte längre är ett aggregat efter eventuella ändringar skall den följa alla regler för god objektorientering som gäller för klasser. Du får inte lägga till fler eller ta bort existerande datamedlemmar i klassen.
- Du får inte använda några loopar eller algoritmen `for_each` för att lösa detta problem. Du skall använda algoritmer från c++ standardbibliotek.
- Du får inte ändra i givna koden utöver klassen Minion.

Tips:

- Du kan skriva till strömmen cout med hjälp av algoritmen `copy` och en `ostream_iterator`, om det finns en överlagring av operatorn `<<` för en klass.

Körexempel:

```
$ ./a.out
Minions to keep vector:
Minion1
Minion3

Minions to fire vector:
Minion2
Minion4
Minion5
```

Uppgift 4 - Hantera en flotta

Kapten Janeway har nyligen införskaffat sig en flotta i spelet Frostgrave. Denna flotta består av olika typer av skepp och dessa skepp värderar hon olika beroende på typ. Du ska designa de klasser som krävs för att det givna huvudprogrammet ska fungera. Det finns två olika specialiceringar av skepp i flottan, `Frigate` och `Transport`. Man skall inte kunna skapa skepp av typen `Ship`, klassen ska alltså vara abstrakt. Följande funktionalitet skall finnas:

- Alla skepp har ett namn och ett prefix. Båda dessa är av typen **std::string**. `Frigate` har också ett antal kanoner. `Transport` har en last. Dessa är av typerna `unsigned short int` respektive `unsigned int`
- Ett skepp skall ha funktionalitet för att returnera en sträng-representation av objektet. Grundbeteendet för alla skepp är att skriva ut prefixet följt av namnet (se körexemplet). `Frigate` gör samma sak men skriver också ut antal kanoner efteråt.
- Alla skepp har möjligheten att returnera sin `worthiness`. Detta är något som inte går att beräkna för basklassen `Ship`. `Transport` beräknar sin worthiness genom att räkna sin `cargo` + 3. `Frigate` beräknar sin worthiness genom att räkna `cannons` * 2.

Huvudprogrammet har i dagsläget ett par buggar relaterade till **slicing** och minnehantering, du behöver ändra koden i huvudprogrammet så att det fungerar som förväntat.

Krav:

- Du skall fördela ansvaret mellan klasserna enligt god objektorientering.
- Du skall inte upprepa kod i onödan.
- Programmet skall köra utan minnesfel, testa med valgrind innan du lämnar in.

Körexempel:

```
$ ./a.out
Fleet :

HMS: Pegasus | Worthiness: 200
HMS: Rotarius | Worthiness: 235
HFR: Santa Rosa - 100 guns | Worthiness: 200
HMS: Daedalus - 150 guns | Worthiness: 300
```