TDP004 - Tenta

2020-03-19

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 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. Tag hjälp av assistent i sal om det inte fungerar.

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 - Polymorfi - Speciella kort

Kapten Janeway spelar spelet Journeys in Middle-earth och i det spelet finns det olika typer av kort. Hon ska nu skapa ett c++ program som låter henne spela spelet med kompisar online. Ditt jobb är att skapa de klasser som krävs för att hantera korten i det spelet.Du ska skapa 3 klasser i en arvsstruktur. `Card`, `ItemCard` och `FateCard`.

`Card` är basklassen och är en abstraktklass. Basklassen lagrar en datamedlem, `name`, och har 2 medlemsfunktioner: `get_description` och `get_cost`. `get_cost` är pure virtual. `get_description` returnerar en beskrivning av kortet, för basklassen är detta endast namnet och kostnaden enligt formatet i körexemplet.

Klassen `ItemCard` utökar basklassen med en ny datamedlem `lore`. Klassen implementerar nu funktionen `get_cost` som för `ItemCard` returnerar `lore`. `ItemCard` utökar också `get_description` så att detta nu returnerar hur mycket Lore kortet har.

Klassen `FateCard` utökar basklassen med två nya datamedlemmar: `successes` och `ability_score`. Klassen implementerar nu funktionen `get_cost` som returnerar ability_score multiplicerat med successes. `get_description` returnerar nu också hur mycket ability_score och antal successes som kortet har.

Du ska också se till att programmet fungerar utan slicing och minnesläckor.

Körexempel:

```
Name: Storm Maker, Cost: 120, Lore: 120
Name: Strider, Cost: 150, Ability Score: 5, Successes: 30
Name: Wooden pipe, Cost: 20, Lore: 20
Name: Durins song, Cost: 100, Ability Score: 2, Successes: 50
```

Uppgift 2 - Operatorer - Öka hälsa

Kapten Picard spelar nu en kampanj av Journeys in Middle-earth med sin besättning. I spelet är det vanligt att ens liv svajar fram och tillbaka pågrund av alla faror som man möter på under resan och därför vill nu kapten Picard ha en klass som motsvarar en livsmätare som kan skrivas ut och ökas.

Du ska nu skapa klassen `Healthbar` som skall ta hand om en spelares hälsa. `Healthbar` skall innehålla funktionalitet för att öka hälsan. `Healthbar` har två datamedlemmar, strängen `name` som innehåller namnet på ägaren av livsmätaren och heltalet `hp` som innehåller värdet på spelarens hälsopoäng. Klassen överlagrar dessutom ett antal operatorer för att addera spelarens hälsopäng (se huvudprogrammet i `uppgift2.cc`). Klassen skall också gå att skriva ut till `cout` med en lämplig operator. Förutom att klassen ska fungera med den givna koden finns följande krav:

- En `Healthbar` ska skapas med enbart ett namn och ett startvärde på hp.
- Ingen koduppprepning är tillåten (återanvänd kod när det är möjligt).
- Den givna koden får inte ändras.
- Alla operatorer som kan vara medlemsoperatorer skall vara det.

Körexempel:

```
Gimlis Health - 5 hp
Gimlis Health - 6 hp
Gimlis Health - 6 hp
Gimlis Health - 7 hp
Gimlis Health - 6 hp
Gimlis Health - 7 hp
Gimlis Health - 7 hp
Gimlis Health - 8 hp
```

Uppgift 3 - Objekt i objekt - Saker på en karta

Archer håller på att skapa en app för Journeys in Middle-earth. Appen ska ha koll på kartbitar i spelet. Du ska skapa klassen **BattleMap**. Klassen representerar externt en kartbit i spelet. En kartbit består av 5 rutor och på dessa rutor kan användaren ställa `GameObject`s. Internt ska alla `GameObject`s sparas i en eller flera lämpliga datastrukturer och fungera som förväntat enligt det givna programmet i `uppgift1.cc`, jämför det givna programmet med körexemplet. En `BattleMap` skapas genom att användaren anger kartbitens namn som en sträng. Spelobjekt skall kunna läggas till på kartbiten genom att skicka in ett objekt och en `char` som motsvarar rutan på kartbiten med medlemsfunktionen `put_object`.

Du ska också skapa klassen **GameObject**. Dessa spelobjekt skapas genom att användaren anger objektets namn och en beskrivning av objektet.

Båda klasserna skall skrivas med korrekt uppdelning i filer. Alla deklarationer skall skrivas i **battlemap.h** och alla implementationer skall skrivas i `battlemap.cc`. Klasserna ska följa regler och principer för god objektorientering.

Körexempel:

```
Map 1:
Square a
Bush Legolas Fireplace

Square b
Table Orc

Square c
Bush

Square d
Bush

Square e
Bush
```

Uppgift 4 - Validering av taggar

Sisco håller på att skapa ett brädspel och har sparat all sin data i xml-format i filer. Tyvärr skriver han ibland fel och pajar filerna och vill därför ha ett enkelt program som validerar att det är korrekt xml.

Din uppgift blir nu att skriva funktionen `validate` som validerar xml-filer. Taggarna består av start taggen `<tag-namn>` och slut-taggen `</tag-namn>`, där tag-namn kan vara en godtycklig sträng som enbart består av bokstäver. Mellan taggarna kan det stå text som innehåller bokstäver, siffror och/eller blanksteg/nyrad.

Funktionen ska läsa filen och plocka ut alla taggar. När en start-tagg hittas ska den läggas till i en lämplig ordnad datastruktur. När en slut-tagg hittas skall denna jämföras med den senaste hittade start-taggen. Om de är lika så ska start-taggen tas bort från behållaren.

Funktionen ska returnera sant om xml-filen är korrekt och falskt om den inte är det. Utöver detta ska funktionen och skriva ut meddelanden till användaren.

Funktionen ska använda minst en algoritm från standardbiblioteket och en databehållare. Dessa skall lösa en icke-trivial del av problemet. Manuella loopar och for_each får endast användas för inläsning från fil.

Körexempel:

```
$ ./a.out
No syntax errors
Tag: '1' is not correct
Tag: 'first' not closed
There are unclosed tags
```

Uppgift 5 - tabell av fiendepekare

Kirk spelar of Journeys in Middle-earth med sin besättning. Ibland blir det många fiender på planen och därför måste du nu skapa ett litet datasystem för att hålla koll på dessa filer.

I den givna koden `uppgift5.cc` finns delar av en datastruktur som innehåller pekare implementerad, klassen heter `EnemyLookup`. Det är en `std::map` där strängar är nycklar och värdena är pekare till instanser av klassen `Enemy`. Klassen `Enemy` representerar en fiende och innehåller namn och hur mycket hälsopoäng en fiende har. `EnemyLookup` har ansvar för minnet för fienderna som den lagrar.

Eftersom klassen har ansvar för minnet så ska det inte gå att kopiera instanser av `EnemyLookup` (det skulle innebära att vi kopierade fienderna också). Det ska däremot gå att flytta `EnemyLookup`. Klassen läcker också i dagsläget minne vilket är något du behöver fixa.

Tips: Kör programmet med valgrind för att testa efter minnesläckor.

```
EnemyLookup:
====
Warg: 15
Goblin: 5
Marauder: 10

Correct, the enemyLookup can not be copied using the assignment operator(=)
Correct, the enemyLookup can not be copied using the copy constructor
```