TDP004 - Tentamen

2019-08-27

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 studentklient.
- 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. Det betyder att för högre betyg måste du på sluttentamen lösa minst 2 uppgifter.

Inloggning

När du blir tillsagd skall du logga in som vanligt med ditt Liu-id och lösenord.

Skrivbordsmiljön

När du kommit in i tentasystemet har du en normal skrivbordsmiljö. Efter en stund kommer även din studentklient att dyka upp automatiskt. En del program kan finnas tillgängliga endast genom att starta dem från en terminal. Om möjligt skall du starta program via ikonerna på

skrivbordet. 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 din studentklient 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 *partiell* tillgång till referenssidorna på `http://www.cppreference.com/` via webbläsaren Chrome. Speciellt går det inte att komma åt language-delen av cppreference (d.v.s. om det står `language` i url:en). 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 studentklienten.

Givna filer

Eventuella givna filer finns i katalogen `given_files` på skrivbordet. 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 din arbetskatalog, vi rekommendrar att skrivbordet är din arbetskatalog. Hur du listar och kopierar filer ska du kunna.

Avslutning

När dina uppgiftsbetyg och ditt slutbetyg i studentklient 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. Lämna inte datorn förrän du ser den vanliga inloggningsskärmen.

Uppgift 1 - Skådespelare

Radiopjäser har flera skådespelare och Kapten Picard är mycket intresserad av att ranka dessa på olika sätt. För att kunna rangordna skådespelarna måste vi dock först kunna representera dem i datorn. Picard har därför gett dig uppdraget att skapa en klass som representerar en skådespelare.

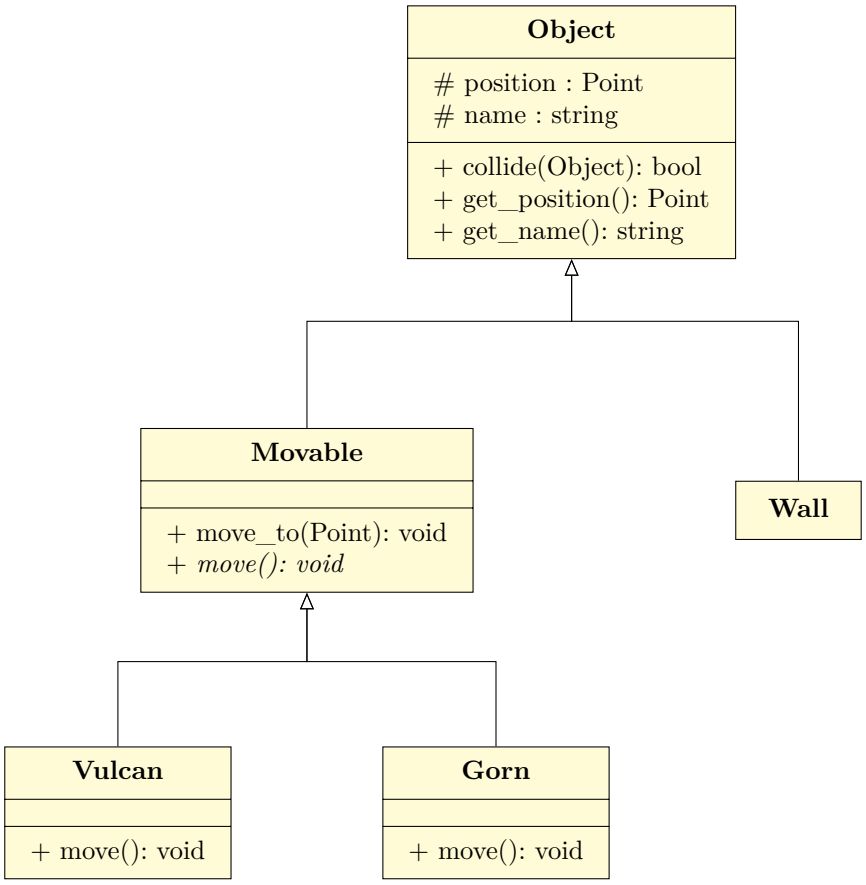
Kopiera filen `given_files/uppgift1.cc` till nuvarande katalog och lägg till klassen `Actor` med den funktionalitet som krävs för att få den givna koden att kompilera. En `Actor` initieras med dess namn samt ett mått på dess berömmelse. När du är klar ska det givna huvudprogrammet kunna sortera samt skriva ut en lista av de tre mest berömda skådespelarna. Du får inte göra någon ändring i den givna koden (utöver att lägga till din klass).

Körexempel:

```
Best 3 actors:
Outaideasstein (10)
Zoe Madeupinton (6)
McGuffinton (2.3)
```

Uppgift 2 - Spelobjekt

Du och dina vänner skall skapa ett sci-fi spel och du har fått ansvar för att skapa de klasser som tar hand om utritandet av spelobjekten. Målet är att spelet ska kunna hantera Vulcaner (observera att detta inte är eldsprutande berg utan en utomjordisk intelligent ras) (klassen `Vulcan`), Gorner (klassen `Gorn`) och väggar (klassen `Wall`). Din uppgift är att skapa en basklass (`Object`) som sparar och låter användaren komma åt objektets position. Klassen `Object` skall även ha en medlemsfunktion (`collide`) som kontrollerar om ett objekt har kolliderat med ett annat. Som standard kommer detta att ske genom att jämföra objektens koordinater men detta skall vara möjligt att överlagra i subklasserna.
`Movable` är en direkt subklass till `Object` som representerar objekt som kan röra på sig i spelet. `Movable` har medlemsfunktionerna `move_to` som sätter positionen och `move` som alla subklasser till `Movable` skall implementera (alla subklasser har med andra ord olika sätt att flytta på sig). Skapa två subklasser till `Movable`; `Gorn` och `Vulcan`. I denna enkla version skall Gorner (`Gorn`) endast röra sig horisontellt (alltså öka med ett i x-led när `move` anropas). När collide anropas blir det kollision om en vägg (`Wall`) är på samma position. Vulcaner (`Vulcan`) skall röra sig vertikalt (alltså öka med ett i y-led när `move` anropas). När collide anropas blir det kollision om en vägg (`Wall`) är på samma position.



Det finns ett kodskelett och testfall givna i `given_files/uppgift2.cc`.

Uppgift 3 - Stacken

Kirk har många böcker. Eftersom han inte tror på flugan med bokhyllor så staplar han alla sina böcker i högar. Han har ett datasystem där han använder datatypen `stack` för att representera högarna av böcker. Systemet är skrivet i C++ och finns i filerna `stack.h` och `stack.cc`. Stacken är en datastruktur som består av `Book`'s. Varje bok består av en sträng för bokens namn och en pekare till nästa Book. Stacken är en datastruktur av typen FIFO (First In First Out). Det innebär att varje bok läggs till först i kedjan och det är den sist tillagda boken som först kommer lämna stacken när man tar ut en bok ur den.

Ditt jobb är att utöka klassen `Stack` så att den kan hantera kopiering, flytt och borttagning på korrekt sätt. Med andra ord behöver du lägga till en `Kopieringskonstruktor`, `Kopieringsoperator`, `Flyttkonstruktor`, `Flyttoperator` och en `Destruktor`. Du behöver göra detta enligt god objektorienterad praxis och med rätt uppdelning mellan filerna. När du är klar behöver du lägga till kod i `uppgift3.cc` (kopiera den från `givna_filer/uppgift3.cc`) för att visa att dina speciella medlemsfunktioner går att köra utan några problem. Den givna koden för stacken får inte ändras och din kod bör fungera så effektivt och felsäkert som möjligt. Kopiering av stacken ska också vara djup så att stackarna inte hänvisar till samma böcker efter kopiering.

Du kan köra din kod med `valgrind ./a.out` efter du kompilerat för att se att den fungerar utan minnesfel. Tänk på att inga minnesfel kommer uppstå om koden i huvudprogrammet inte testar den delen av klassen. Det är därför viktigt att du modifierar huvudprogrammet så att alla speciella medlemsfunktioner körs.

Uppgift 4 - Synonymer

Du har fått en beställning på ett program som skall ta emot en text och byter ut varje ord med en synonym om det är möjligt. Denna gången[1] vet kunden om vad en synonym är. Kunden har gett dig en fil `given_files/SYNONYMS` som innehåller en lista av synonymer. Varje rad i filen är skriven på följande format:
`<word to be replaced> = <word to replace with>`
där varje förekommande ord på vänster sida av listan ska bytas ut mot det som står på den högra sidan. Ditt program skall göra följande:

1. Öppna filen `SYNONYMS` för inläsning.
2. Läs varje rad i `SYNONYMS` och separera raden på `=`, lägg in ordparet (det vänstra och högra ordet) i en lämplig behållare med namnet `synonyms`.
3. Läs in en text ord för ord från `cin` till en lämplig behållare med namnet `words`. Om ett ord finns med i `synonyms` ska ordet bytas ut mot motsvarande värde i `synonyms`, annars skall ordet lämnas som det är.
4. Skriv ut alla ord i `words` separerade med mellanslag.

Det finns ett kodskelett och testfall givna i `given_files/uppgift4.cc`.

[1] Det fanns en liknande uppgift på tentamen som gavs 2019-04-26

Uppgift 5 - Rövarspråket

Rövarspråket är ett kodspråk som används bland barn. Det följer en enkel regel; efter varje konsonant lägger man till ett `o` följt av samma konsonant igen (d.v.s. konsonanter såsom `f` ersätts med `fof`). Exempel: `kaffe` blir `kokafoffofe` och `oavsett` blir `oavovsosetottot`.
I denna uppgift ska du skapa ett program som:

1. Läser in ord från `cin` till en vektor tills filslut (`ctrl-D`)
2. Konverterar alla ord i vektorn till rövarspråket (lägg denna funktionalitet i en funktion)
3. Skriver ut alla ord på rövarspråket (separerade med ett mellanslag)

Du får inte använda några manuella loopar, utan du måste använda algoritmer från standardbiblioteket. Du får inte heller använda `std::for_each`. Du får inte heller använda fullständiguppräkning för att ta reda på om en bokstav är en vokal eller konsonant. Använd istället den givna strängen `vowels` för att se om tecknet finns med där. **Notera:** Ditt program behöver **inte** hantera å, ä eller ö.

Ni hittar ett kodskelett i `givna_filer/uppgift5.cc`

Körexempel (fetstilt är användarinmatning)

```
Mata in din text: vi ska koka kaffe imorgon
vovi soskoka kokokoka kokafoffofe imomororgogonon

Mata in din text: det ska vara kul att skriva c++
dodetot soskoka vovarora kokulol atottot soskokrorivova coc++
```