TDP004 - Tentamen

2019-01-16

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

Innan du loggar in ska du välja "Exam system" från sessionsmenyn i nedre vänstra hörnet av inloggningsrutan. Därpå loggar du in med dina LiU inloggningsuppgifter. Du kommer nu in i tentainloggningen och ska börja med att välja språk. Ta den svenska flaggan så blir det Svenska (valet spelar ingen roll för dig som kan båda språken). Följ instruktionerna på skärmen så långt det går tills du ska mata in ett engångslösenord. Tag fram ditt LiU-kort och visa det för assistent eller vakt i sal för att få detta lösenord.

Skrivbordsmiljön

När du kommit in i tentasystemet har du en normal skrivbordsmiljö med grön skrivbordsbakgrund. Efter en stund kommer även din studentklient 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 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-delenav 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`. 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. Hemkatalogen heter alltid `/home/student_tilde` om du undrar.

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 tryck på knappen märkt "Exit" i menyn längst ner på skärmen och välj "ok". Vänta ett tag och tryck sedan på knappen "Avsluta tentamen" när det är möjligt. När detta är gjort är det omöjligt att logga in igen. Lämna inte datorn förrän du ser den vanliga inloggningsskärmen. Anmäl till assistent eller vakt om inloggningsskärmen inte dyker upp inom en minut så åtgärdar vi problemet.

Uppgift 1 - Viskleken (personligheter)

Notera att alla uppgifter är fristående från varandra.

I leken ”Viskleken” börjar en person genom att viska en mening i någon annans öra. Denna ska sedan viska det vidare till någon annan och efter flera led ska den sista berätta vad den hört. I denna uppgift ska du, genom att representera olika personligheter i en klasshierarki, simulera en kort session av viskleken.

Klasshierarkin består av basklassen `Personality` med specialiseringarna `Forgetful` och `Confabulator`. Klasserna ska ha funktioner för att processera en viskning enligt följande:

- `Forgetful` glömmer upprepa N slumpade ord i viskningen.
- `Confabulator` stoppar in ordet “super” före det längsta ordet.

Parametern N anges när klasserna skapas i huvudprogrammet. Huvudprogrammet frågar användaren om en viskning, låter den processas av varje personlighet, och skriver ut resultatet. Viskningen består av ett eller flera ord. I ditt program får du representera viskningen med vilken datatyp du vill. En `std::vector<string>` rekommenderas. Att senare lägga till en person i visksekvensen ska gå att göra utan att skriva ett nytt anrop till funktionen som processar en viskning.

I denna uppgift godkänner vi att din slumpgenerator ligger som en global variabel. Du ska använda C++ slumpning (dvs *inte* `rand`).

Det finns ett påbörjat förslag till ett testprogram i `given_files/uppgift1.cc`. Du måste ta hand om minneshantering och initiering av vektorn själv. Vi tillåter att du använder smartpekare.

Uppgift 2 - Viskleken (ordklasser)

Notera att alla uppgifter är fristående från varandra.

I leken ”Viskleken” börjar en person genom att viska en mening i någon annans öra. Denna ska sedan viska det vidare till någon annan och efter flera led ska den sista berätta vad den hört. I föregående uppgift skapade du olika personligheter. För att lättare kunna skapa olika personligheter vore det bra med tillgång till en ordlista som kan hjälpa oss avgöra om ett ord tillhör en viss ordklass. I denna uppgift ska du skapa en klass för en sådan ordlista.

Ordlistan ska ha följande funktionalitet:

- `is_adjective` ger sant om ett ord är ett adjektiv.
- `is_noun` ger sant om ett ord är ett substantiv.
- `is_verb` ger sant om ett ord är ett verb.

När en ordlista skapas ska den automatiskt läsa in ord från filen “ORD.TXT”. Varje rad i filen innehåller ett ord följt av ordets ordklass. Samma ord kan finnas på flera rader i filen med olika ordklass (och betydelse). Alla ord är skrivna med gemener (små bokstäver). Exempel:

```
springa verb
löpa verb
springa substantiv
skreva substantiv
```

Ditt huvudprogram ska skapa en ordlista, be användaren om ett ord, och skriva ut om ordet tillhör respektive ordklasser (verb, substantiv, adjektiv).

Körexempel (användarinmatning i fet stil)

```
Mata in ett ord:
springa
springa är ett verb
springa är ett substantiv
```

Uppgift 3 - Viskleken (standardbiblioteket)

Notera att alla uppgifter är fristående från varandra.

I leken ”Viskleken” börjar en person genom att viska en mening i någon annans öra. Denna ska sedan viska det vidare till någon annan och efter flera led ska den sista berätta vad den hört. I uppgift 1 skapade du olika enkla personligheter. I denna uppgift ska vi implementera en knepig personlighet. Alla steg ska lösas med hjälp av standardbiblioteket STL och dess algoritmer (inga egna loopar).

1. Läs in en rad med ord till en vanlig sträng.
2. Kopiera varje ord från strängen till en std::vector.
3. Byt plats på det kortaste och längsta ordet.
4. Infoga “Super” framför alla ord med inledande versal (stor bokstav).
5. Ta bort alla ord som är krångliga. Krångliga ord innehåller något tecken som inte är en bokstav.

Skriv ut alla orden före och efter varje steg (totalt 4 utskrifter). Programmet ska alltid avsluta kontrollerat, även för blanka rader.

Körexempel (användarinmatning i fet stil)

```
Hej hopp fallera nu ar julen slut slut!
Hej hopp fallera nu ar julen slut slut!
Hej hopp nu fallera ar julen slut slut!
SuperHej hopp nu fallera ar julen slut slut!
SuperHej hopp nu fallera ar julen slut
```

Uppgift 4 - Viskleken (fram och åter)

Notera att alla uppgifter är fristående från varandra.

I en variant på viskleken viskar vi meningen hela vägen tillbaka till den första personen. Då måste en person kunna viska både framåt och tillbaka. I denna uppgift ska du skapa en klass `Whisper_List` för en kö av personer som ska viska först framåt och sedan hela vägen tillbaka i omvänd ordning. Din `Whisper_List` ska kunna:

- Skapas och då bli alldeles tom.
- Fyllas på med en person som ställs sist.
- Flyttas med en flyttkonstruktor och flyttilldelningsoperator.
- Itereras i en sekvens från början till slut och tillbaka igen.

Det ska vara omöjligt att kopiera `Whisper_List`.

`Whisper_List` ska implementeras som en dubbellänkad lista uppbyggd av den givna nod-typen.

Iterering genom `Whisper_List` ska börja på första elementet och iterera till sista. När itereringen har nått sista elementet ska den fortsätta från sista tillbaka till första elementet igen. Efter det är klart ska funktionen `has_next` returnera `false`.

Följande funktioner ska finnas i `Whisper_List`:

- `insert` ska ta en sträng och stoppa den längst bak i listan.
- `get_current` ska returnera det nuvarande namnet i itereringen.
- `reset` ska förbereda klassen för itereringen från början.
- `has_next` ska returnera `true` så länge itereringen inte har kommit tillbaka till början.
- `next` ska stega till nästa värde i itereringen, denna funktion ska hantera att itereringen vänder vid slutet av listan.

Korrekt minneshantering är ett krav, inga minnesläckor får förekomma i programmet. Det finns en `Node` klass, pseudokod för `insert` funktionen och ett huvudprogram i `given_files/uppgift4.cc` som ska fungera utan några ändringar.

Tips: `Whisper_List` kommer behöva åtminstone datamedlemmarna `first` och `last` som är av typen `Node*`. Notera att det kan behövas att du lägger till datamedlemmar för att hantera itereringen.

Uppgift 5 - Viskleken (fram och åter igen)

Notera att alla uppgifter är fristående från varandra.

I en variant på viskleken viskar vi meningen hela vägen tillbaka till den första personen. Då måste en person kunna viska både framåt och tillbaka. I denna uppgift ska du skapa en klass `Whisper_Vector` för en kö av personer som ska viska först framåt och sedan hela vägen tillbaka i omvänd ordning. Din `Whisper_Vector` ska kunna:

- Skapas och då bli alldeles tom.
- Fyllas på med en person som ställs sist.
- Itereras i en sekvens från början till slut och tillbaka igen.

`Whisper_Vector` ska implementeras som en klass med en `std::vector` som datamedlem. Iterering genom `Whisper_Vector` ska börja på första elementet och iterera till sista. När itereringen har nått sista elementet ska den fortsätta från sista tillbaka till första elementet igen. Efter det är klart ska funktionen `has_next` returnera `false`.

Följande funktioner ska finnas i `Whisper_Vector`:

- `insert` ska ta en sträng och stoppa den längst bak i listan.
- `get_current` ska returnera det nuvarande namnet i itereringen.
- `reset` ska förbereda klassen för itereringen från början.
- `has_next` ska returnera `true` så länge itereringen inte har kommit tillbaka till början.
- `next` ska stega till nästa värde i itereringen, denna funktion ska hantera att itereringen vänder vid slutet av listan.

Det finns ett huvudprogram i `given_files/uppgift5.cc` som ska fungera utan några ändringar.