TDP004 - Tentamen

2015-04-08

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 eller rökpaus 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. C++ Primer 5th ed.) En A4-sida 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. *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**
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 1: 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 4 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. 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

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 flagga som ser minst Engelsk ut 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 (Mate-session i Linux Mint) med grön skrivbordsbakgrund. 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++11` används för att kompilera med "alla" varningar *som fel*.
`w++11` används för att kompilera med "alla" varningar. **Rekommenderas.**
`g++11` används för att kompilera `utan` varningar.
`gccfilter e++11` används för att köra `e++11` genom `gccfilter`.
`gccfilter w++11` används för att köra `w++11` genom `gccfilter`.
`gccfilter g++11` används för att köra `g++11` genom `gccfilter`.
`valgrind --tool=memcheck` används för att leta minnesläckor.
`vim` kommer du ut ur genom kommandot `:q!`

C++ referenssidor

På tentan har du *experimentiell* tillgång till referenssidorna på `http://www.cpluplus.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. Hemkatalogen heter alltid `/home/student_tilde` om du undrar.

Avslutning

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 med blå bakgrund. Anmäl till assistent eller vakt om inloggningsskärmen inte dyker upp inom en minut så åtgärdar vi problemet.

Uppgift 1 Logrotering av filer

Inledning

Logfiler har en tendens att växa sig mycket stora. Därför tar man ibland en kopia av aktuell logfil och påbörjar en ny tom fil. Då detta görs upprepade gånger råkar vi snart på problem med namngivningen. Vi vill att logfilerna ska numreras så att äldst fil har högst nummer. Ska vi kopiera `log.txt` till `log1.txt` kan det hända att `log1.txt` redan finns. Då måste vi först kopiera `log1.txt` till `log2.txt` och kanske `log2.txt` till `log3.txt` och så vidare. På detta sätt roteras filnamnen.

Funktionella krav

Du skall skriva ett program som skriver ut de terminalkommandon som behövs för att utföra logrotering av en viss fil. Programmet skall fungera enligt körexempel och ickefunktionella krav.

Ickefunktionella krav

- Ett logfilnamn skall representeras av en klass `Logfilename`. Klassen ska enbart ha publik funktionalitet enligt följande lista:
  - En defaultkonstruktor.
  - Medlemsfunktion(er) som returnerar filnamnet. Filnamnet ska returneras med numrering före punkten om ett heltal anges som argument. Utan argument, inget nummer.
  - En inmatningsoperator för inläsning av filnamnet. Inmatningen behöver inte felkontroll och kan utgå från att alla filnamn har exakt en punkt före filändelsen och inga blanksteg i namnet.
  - En funktion som skriver ut de kommandon som behövs för att rotera filen i ett antal steg.
- Huvudprogrammet skall använda samtliga delar av klassen.

Tips och begränsningar

- Inläsning och privata delar av klassen kan utformas efter eget tycke för att få till en lösning, men enkelhet premieras.
- Programmet behöver bara fungera korrekt för en högsta numrering som är ett eller högre.

Körexempel (användarinmatning fetstilt)

```
What is the file name? apache.log
What is the new highest numbered file? 5
The following commands will rotate the files:
mv "apache4.log" "apache5.log"
mv "apache3.log" "apache4.log"
mv "apache2.log" "apache3.log"
mv "apache1.log" "apache2.log"
mv "apache.log" "apache1.log"

What is the file name? security_audit_log.txt
What is the new highest numbered file? 3
The following commands will rotate the files:
mv "security_audit_log2.txt" "security_audit_log3.txt"
mv "security_audit_log1.txt" "security_audit_log2.txt"
mv "security_audit_log.txt" "security_audit_log1.txt"

What is the file name? error.xml
What is the new highest numbered file? 1
The following commands will rotate the files:
mv "error.xml" "error1.xml"
```

Uppgift 2 Monty Hall problemet

Inledning

I en tävling ställs den tävlande framför tre stängda dörrar. Bakom två av dörrarna finns en get och bakom en dörr finns en bil. Den tävlande får nu välja en dörr (utan att öppna den). Därpå öppnar tävlingsledaren en av de båda andra dörrarna, hen väljer alltid en dörr med en get. Slutligen får den tävlande möjlighet att antingen stå fast vid sitt val av dörr, eller ändra sig. Om vi antar att den tävlande har som mål att välja dörren med bilen, är det då bättre att ändra sig?

Funktionella krav

Eftersom det är bra mycket besvärligare att tänka och använda sannolikhetskalkyl än att programmera skall du i denna uppgift simulera problemet en miljon gånger. Ditt program ska sedan skriva ut hur stor sannolikhet den tävlande har att hitta bilen om hen byter dörr. Simuleringen av en omgång ska gå till ungefär så här:

1. Skapa tre numrerade dörrar.
2. Slumpa ut bilen och två getter bakom dörrarna.
3. Låt den tävlande välja dörr 1.
4. Låt tävlingsledaren välja en av de båda andra dörrarna som döljer en get.
5. Låt den tävlande byta dörr.
6. Kontrollera om den tävlande valde dörren med bilen detta försök.

Ickefunktionella krav

- Programmet ska utföra en miljon försök enligt ovan modell. Du är fri att välja hur olika delar ska representeras och fri att göra smärre förenklingar inom varje punkt.
- Du ska använda `std::random_device` och `std::uniform_int_distribution` för att få fram slumptal.
- Du ska använda en `std::array` eller `std::vector` för att lagra vad som finns bakom varje dörr. Detta underlättar framförallt vid slumpningen.
- Resultatet ska skrivas ut i procent med en decimals noggrannhet.

Tips och begränsningar

- Eftersom rätt svar inte är givet, fundera på hur många gånger av hundra den tävlande från början kommer välja dörren som döljer bilen. Fundera sedan på hur många gånger av hundra dörrbyte kommer resultera i att rätt val “förloras”. Övriga fall är vinnande, det finns ju bara en dörr att byta till.

Körexempel (OBS! du bör få fram en annan procentsats)

```
$ ./a.out
We win the car 50.0% of the time if we switch.
```

Uppgift 3 Kryp (given_files/test_kryp.cc, given_files/kryp.h)

Inledning

En kryp börjar som en tom hög. Nya värden läggs alltid till underst i högen (till skillnad från en stack, som ju placerar nya värden överst). När värden så småningom skall tas bort tar man likaså det understa värdet i högen (också detta tvärtemot en stack, som tar det översta).

Funktionella krav

Skriv en klass som representerar en kryp med operationerna för att lägga till en sträng i krypen, plocka ut en sträng från krypen, och kontrollera om krypen är tom.

Ickefunktionella krav

1. Din kryp ska byggas upp av krypelement (här kallar vi dem myror) där vardera myra pekar på nästa myra.
2. Din kryp ska ha korrekt minneshantering.
3. Din kryp ska använda god abstraktion och inkapsling.
4. Operationerna på krypen ska heta *push* för insättning, *pop* för borttagning och *empty* för att kontrollera om krypen är tom.
5. Nyckelordet `const` ska användas där det är lämpligt.
6. Din klass ska skrivas med korrekt inkluderings- och implementationsfil.
7. Kopieringskonstruktor, tilldelningsoperator, movekonstruktor och movetilldelning ska vara korrekt implementerade eller åtgärdade på sådant sätt att användning av dem ger kompileringsfel.
8. Det givna huvudprogrammet ska fungera utan modifikation.
9. Utskrifterna ska matcha körexemplen till fullo.

Tips och begränsningar

- Att ta det understa i en hög är mot allt förnuft eftersom högen riskerar rasa. Dock har vi inga sådana bekymmer i datorn, så då är det kanske inte en så dum idé ändå.
- Filen `given_files/kryp.h` innehåller en början på klassdefinition som dock behöver kompletteras och modifieras för att uppfylla vissa av ovan krav.
- Tänk på att skicka med alla filer som behövs vid kompilering när du är klar.
- Det är inte krav på undantagshantering, men utskrifterna ska matcha körexemplen till fullo.

Körexempel (användarinmatning kursivt, filslut fetstilt)

```
tom kryp
lagger saker underst
Ctrl-D
underst saker lagger tom kryp
tom kryp
```

Uppgift 4 Rättning av tipspromenad

Inledning

Efter en virtuell tipspromenad har du fått in svar från varje tävlande i form av en sträng som anger 1X2 för varje fråga. T.ex. anger strängen "1122-X21XX" att den tävlande besvarade de båda första frågorna med svarsalternativ 1, den femte besvarades inte och för de båda sista valdes svaralternativ X.

Funktionella krav

Den rätta raden inklusive en förklarande mening av rätt svar finns lagrat på filen `given_files/quiz.txt`. Formatet är radbaserat och enkelt. Först på varje rad kommer rätt svarsalternativ (ingen skillnad görs mellan stora och små bokstäver), sedan ett semikolon, och sist en förklaring av rätt svar i text. Detta upprepas med en rad per fråga.

Skriv ett program som läser in en svarsrad från användaren och sedan skriver ut hur många rätt raden gav. Viss felkontroll ska utföras enligt körexempel. Filen med rätt svar ska anges som argument till main (kommandoradsargument) så programmet enkelt kan användas med många olika filer.

Ickefunktionella krav

1. Ingen skillnad görs mellan stora och små bokstäver i svarsalternativen, headerfilen `cctype` kan vara användbar.
2. Utskrifterna ska matcha körexemplen till fullo.

Körexempel (kommandorad fetstilad, användarinmatning kursiv)

```
$ ./a.out
ERROR: solution file argument missing.

$ ./a.out no_such_file
ERROR: solution file could not be opened.

$ ./a.out given_files/quiz.txt
Please enter given answer: 1111
WARNING: too few questions answered.
1 point(s).

$ ./a.out given_files/quiz.txt
Please enter given answer: 11111111111111111111
WARNING: too many questions answered.
5 point(s).

$ ./a.out given_files/quiz.txt
Please enter given answer: 2222222222222
4 point(s).

$ ./a.out given_files/quiz.txt
Please enter given answer: 1x2xx121122x1
13 point(s).
```

Uppgift 5 Justify text

Inledning

Ibland vill man formatera text så både vänster och höger kant blir rak och fin. Detta är besvärligare än man först kan tro, och därför skall vi skapa en klass som gör det lite lättare.

Funktionella krav

Programmet skall läsa in en rad i taget från användaren och därefter skriva ut raden med exakt 80 teckens bredd. Saknas tecken ska extra blanksteg fyllas på mellan ord tills raden är lång nog.

Ickefunktionella krav

1. Problemet ska lösas med klassen `Justify`. Klassen ska ha en konstruktor som tar emot antalet ordmellanrum på raden (antalet ord minus ett), och totala antalet blanksteg som dessa mellanrum måste använda. Privat i klassen skapas en lämplig behållare som kan lagra hur många blanksteg det ska finnas i varje ordmellanrum.

2. Klassen ska ha en funktion som hämtar ut antalet blanka tecken som behövs för ett visst mellanrum.

3. Det skall bara krävas en enkel ändring för att byta radlängd.

Tips och begränsningar

- För att enkelt avgöra hur många blanksteg det ska finnas i varje ordmellanrum rekommenderas du att börja med noll blanksteg på varje position i mellanrumsbehållaren och sedan för varje blanksteg som behövs öka varje position i mellanrumsbehållaren med ett blanksteg i taget. När du når slutet på behållaren börjar du om från början. Detta görs förstås i klassen innan något skrivs ut.

- Några mellanrum kommer bli ett steg bredare än andra. För att sprida ut dessa så inte alla breda gap hamnar i början eller slutet av raden rekommenderas du att blanda mellanrumsbehållaren med t.ex. `std::random_shuffle`.

- Vi förutsätter att en inmatad rad innehåller minst två ord med som mest 79 tecken totalt.

Körexempel (nedkortat med användarinmatning omdireigeras från fil)

```
$ ./a.out < lorem.txt
Lorem  ipsum  dolor sit amet,  consectetuer  adipiscing  elit,  sed diam nonummy
nibh euismod tincidunt ut laoreet  dolore  magna aliquam erat volutpat. Ut  wisi
enim ad  minim veniam, quis nostrud  exerci tation ullamcorper suscipit lobortis
.
.
consuetudium  lectorum.  Mirum  est  notare  quam  littera  gothica,  quam  nunc
putamus  parum claram,  anteposuerit litterarum  formas humanitatis  per seacula
quarta  decima et quinta decima. Eodem  modo typi, qui nunc nobis videntur parum
clari,            fiant             sollemnes            in             futurum.
1
2                                                                              2
3                                      3                                       3
4                          4                         4                         4
5                   5                   5                   5                  5
6              6               6               6               6               6
7            7             7            7            7            7            7
8           8          8          8          8          8          8           8
9         9         9         9         9        9         9         9         9
a       a        a        a        a        a        a        a       a        a
```