TDP004 - (För)Tentamen

2015-12-07

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 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, 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 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 (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

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.

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

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 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 Scrabble

Inledning

På filen `given_files/scrabble.cc` finns det lite kod given för att hantera poängberäkning i spelet Scrabble. I Scrabble ska spelaren hitta ord som ger så mycket poäng som möjligt. Ordets poäng är summan av varje bokstavs poäng. De som skapat denna kod har valt ett sätt att lagra informationen om vad ett visst tecken är värt på ett sätt som visserligen är enkelt att skriva och ändra, men som tyvärr försvårar sökning av ett visst tecken.

Funktionella krav

Din uppgift är att implementera poängberäkningen av inmatade engelska ord. Användaren matar in ett ord per rad och avslutar inmatningen med Ctrl-D på tom rad. Inmatade ord kan bestå av både gemener och versaler. En bokstav ger samma poäng oavsett om den är gemen eller versal. Körexemplet visar hur det ska fungera.

Ickefunktionella krav

- Det ska räcka att ändra i den givna datastrukturen för att ändra vilken poäng en viss bokstav ger.
- Den givna datastrukturen får enbart kopieras en gång.
- Din lösning ska använda minst en STL-algoritm (t.ex. `std::transform` för att göra om ett ord till en poängvektor, och `std::accumulate` för att summera poängvektorn).
- Din lösning ska ha minst en lämplig funktion utöver `main`.
- Utdata ska matcha körexemplen till fullo.

Tips och begränsningar

- Det är tillåtet att programmet kopierar över den givna datastrukturen till en datastruktur som förenklar sökning.
- Inmatade ord innehåller enbart tecken som finns representerade i den givna datastrukturen.

Körexempel (användarinmatning i fet stil)

```
$ ./a.out
Enter words:
Apple
"Apple" gives 9 points.
Xyjkyx
"Xyjkyx" gives 37 points.
XKCD
"XKCD" gives 18 points.
oxyphenbutazone
"oxyphenbutazone" gives 41 points.
XKCDQuiz
"XKCDQuiz" gives 40 points.
```

Uppgift 2 Datum

Inledning

I ett projekt finns funktioner för att hantera datum. Dessa funktioner hittar du i den givna filen `given_files/date.cc`. Nu har projektgruppen bestämt sig för att skriva programmet objektorienterat. Det har blivit din uppgift att skriva om datumfunktionerna till en klass som representerar ett datum samt komplettera med klassen funktionen `next_date`.

Funktionella krav

Funktionen `next_date()` ska givet ett datum räkna fram datumet som representerar dagen efter det givna datumet. Funktionen tar inga parametrar och har inget returvärde, utan arbetar på sin klassinstans. Din datumklass ska stödja skottår, dvs dagen efter 2012-02-28 är 2012-02-29 medan dagen efter 2013-02-28 är 2013-03-01.

När du gjort klassen ska du även skapa ett program som låter användaren mata in ett datum. Programmet ska kontrollera så detta datum är korrekt och sedan skriva ut det datum som inträffar 10000 dagar efter det angivna datumet enligt nedanstående körexempel.

Ickefunktionella krav

- Klassen ska vara uppdelad i korrekt deklarations- och implementationsfil.
- Klassen ska ha alla funktioner som finns givna och dessutom funktionen `next_date()`.
- Klassen ska vara väl inkapslad (dvs allt kanske inte är lämpligt att ha "public").
- Utskrifterna ska matcha körexemplen till fullo, med två siffror för månad och dag.

Tips och begränsningar

- Du kan utgår från att användaren matar in enbart siffror separerade med två enskilda bindestreck.
- Det är tillåtet att lägga till medlemsfunktioner eller konstruktorer utöver det som finns i givna koden så länge klassen är väl felhanterad och inkapslad.

Körexempel (användarinmatning i fet stil)

```
$ ./a.out
Enter a date: 2012-02-29
10000 days later: 2039-07-17

$ ./a.out
Enter a date: 2013-13-02
Invalid date, enter another date: 2013-12-42
Invalid date, enter another date: 1900-02-29
Invalid date, enter another date: 1987-04-13
10000 days later: 2014-08-29
```

Uppgift 3 Valutasummering

Inledning

På filen `given_files/exchange_rates.txt` finns växlingskurser som beskriver hur många svenska kronor man får för en enhet av given valuta. Exempelvis ger en KRV (Sydkoreansk won) 0.007069 kronor. Data är taget ifrån riksbanken för datumet 2015-09-25.

Funktionella krav

Din uppgift är att skapa ett program som ber användaren mata in ett antal belopp. När användaren är klar med inmatningen (trycker Ctrl-D på tom rad) ska programmet skriva ut summan av alla belopp uttryckt i svenska kronor med två decimaler.

Användaren kan mata in belopp på formatet "värde NNN" som ska tolkas som ett värde i valuta NNN. Användaren kan även välja att endast mata in ett värde utan valuta. Valutan antas då vara svenska kronor (SEK).

Om användaren matar in en valuta som inte finns med i filen ska *programmet* upptäcka detta och avslutas eller krascha med ett felmeddelande. Att operativsystemet terminerar programmet med "segmentation fault" är alltså fortfarande fel, medan t.ex. något liknande körexemplet är okej. Medellandet behöver i detta fall inte vara användaranpassat.

Ickefunktionella krav

1. Namnet på filen med växlingskurser ska ges på kommandoraden, med tillämpliga felkontroller.
2. Lösningen ska använda en lämplig STL-container.
3. Felaktiga användarinmatningar ska ignoreras helt av programmet.
4. Utdata ska matcha körexemplen till fullo där inte annat sagts ovan.

Körexempel (användarinmatning i fet stil)

```
$ ./a.out
Usage: ./a.out FILE
$ ./a.out ../given_files/exchange_rates.txt och-mer
Usage: ./a.out FILE

$ ./a.out given_files/exchange_rates.txt
Enter amounts (finish by Ctrl-D):
12.3
3.3 AUD
hejhopp
12.6 USD
503 RUB
Total amount in SEK: 202.43

$ ./a.out given_files/exchange_rates.txt
Enter amounts (finish by Ctrl-D):
3.5 SEK
12.3 LOL
terminate called after throwing an instance of 'std::out_of_range'
```

Uppgift 4 Räkneprogram

Inledning

I `given_files/calculator.cc` finns början på ett räkneprogram.

Programmet låter användaren mata in flyttalsvärden, operatorn + eller operatorn * i blandad ordning. Programmet läser in uttryck genom att lägga alla värden (operander) som kommer in på hög. När en operator (+ eller *) kommer in så tas de två översta uttrycken bort från högen, operatorn skapas med dessa två uttryck som operander, och nyskapade operatorn läggs överst i högen.

Funktionella krav

När inmatningen är slut beräknas slutresultatet genom att utvärdera uttrycket som ligger överst i högen. Hur utvärderingen sker beror lite på vad det är som ligger överst. Om det är en operand returnerar den helt enkelt sitt värde. Om det är en operator beräknas först värdet av operatorns två operanduttryck. Sedan adderas eller multipliceras dessa två värden beroende på vilken av operatorerna det är och resultatet returneras.

Ickefunktionella krav

Du ska fylla på programmet med de klasser som saknas för att det ska fungera. Det ska finnas en basklass `Expr` och subklasserna `Operand`, `Operator`, `Plus` och `Mult`. En operand behöver bara lagra sitt värde, medan en operator behöver lagra en pekare till vardera operanduttryck. Varje klass behöver implementera sin version av den polymorfa beräkningsfunktionen `evaluate()`. Din lösning får inte läcka minne.

Tips och begränsningar

- Filuppdelning krävs inte, allt kan skrivas i en fil.
- I körexemplen skrivs även det beräknade uttrycket ut så som det skulle se ut enligt vanligt skrivsätt (med alla parenteser). **OBS! Detta behöver inte ditt program klara.**
- Det finns inte behov av att ändra huvudprogrammet om klasserna implementeras korrekt.
- Körexempel ett kommer att lägga tre operander på högen, sedan skapa operatorn “gånger” med “4” och “8” som operanduttryck, och slutligen skapa operatorn “plus” med “gånger” och “2” som operanduttryck.

  Operatorn “plus” kommer därmed hamna som det enda (översta) på högen. När “plus” utvärders adderas utvärderingen av dess två operanduttryck. I det ena får vi direkt “2” och i det andra får vi “32” eftersom det är uttrycket “gånger” med operanderna “8” och “4”. Summan blir alltså “34”.

Körexempel (användarinmatning i fet stil)

```
$ ./a.out
2 4 8 * +
(2+(4*8))
Answer: 34
```

```
$ ./a.out
2 4 8 + *
(2*(4+8))
Answer: 24
```

```
$ ./a.out
2 4 * 8 +
((2*4)+8)
Answer: 16
```

```
$ ./a.out
2 4 + 8 *
((2+4)*8)
Answer: 48
```