TDP004 - (För)Tentamen

2014-12-04

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 tentamnenstidens 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, 3 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.
`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 Biljettsläpp

Inledning och funktionella krav

I denna uppgift ska vi simulera ett biljettsläpp. Detta går till så att huvudprogrammet får representera "kassan" som betjänar en person per minut. Nya personer anländer varje minut genom att användaren av programmet matar in personernas förnamn på en rad. Olika namn separeras med blanksteg.

Varje inmatad rad representerar en minuts simulerad tid. Programmet ska skriva ut vem som betjänas efter varje inmatad rad. En tom rad symboliserar att inga nya köande anlände denna minut. Om kön av väntande biljettköpare blir tom skrivs detta ut enligt körexempel.

Användaren avslutar inmatningen genom att trycka `Ctrl-D` på en tom rad. Detta betyder att biljetterna är slutsålda och de som är kvar i kön får gå hem. Programmet ska skriva ut namnet på alla som får gå hem utan biljett enligt körexempel.

Ickefunktionella krav

- För att representera en väntande person ska du använda en instans av klassen `Person` (som du skapar). Det ska inte gå att skapa personer utan att ange åtminstone förnamn.
- Din klass ska nyttja inkapsling så långt det går.
- Utskrifterna ska matcha körexemplen till fullo.

Tips och begränsningar

- Allt som inte är blanka tecken räknas som ett giltigt förnamn.
- Du får använda valfri STL-behållare för att representera kön av väntande biljettköpare.
- Din klass blir mycket liten och kan därför skrivas direkt i samma fil som huvudprogrammet.

Körexempel (användarinmatning kursivt, filslut fetstilt)

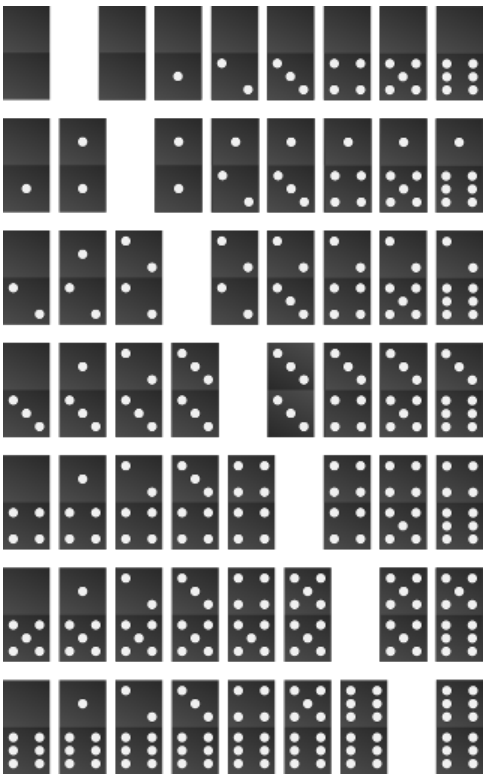
```
Biljetter! Kom och Köp!
Astrid Adam Bertil Eva
Astrid betjänas nu.
Fredrik
Adam betjänas nu.

Bertil betjänas nu.

Eva betjänas nu.

Fredrik betjänas nu.

Kön är tom, ingen betjänas nu.
Greta Göran Zlatan
Greta betjänas nu.
Ctrl-D
Följande personer fick ingen biljett:
Göran
Zlatan
```

Uppgift 2 Domino

Inledning

Ett dominoset består av avlånga brickor, dubbelt så långa som breda med ett streck på mitten som delar en bricka i två lika halvor. På varje halva står en siffra mellan noll och största siffran i setet, och varje kombination siffror ska förekomma exakt en gång i varje helt set. Ett set med brickor brukar namnges efter den bricka med största siffran på båda halvor. Det vanligaste setet "double-six" har således siffror i intervallet [0..6].

Antalet brickor i ett set kan räknas fram utifrån högsta siffran. I fallet med "double-six" kan siffrorna [0..6] kombineras på $(7 * 8)/2 = 28$ unika sätt, det blir alltså 28 olika brickor. *Ett korrekt dominoset innehåller alltså alltid ett givet antal brickor som alla är unika.* Då 28 inte är så många brickor finns set med högre största siffra, upp till "double-twentyone" med $(22 * 23)/2 = 253$ brickor.



Figur 1: Två "double-six" set, $(7 * 8)/2$ brickor per set

Figuren visar två uppsättningar "double-six" set vända och hopsatta längs diagonalen så du ser hur man kan komma fram till totala antalet brickor i ett set givet att man vet högsta siffran. Lägsta siffran är alltid noll. För "double-five" kan du tänka dig att nedersta raden tas bort, och kolumnen längst till höger tas bort. För "double-seven" lägger du till en rad och kolumn. I figuren kan du även se hur ett visst set kan genereras med loopar om du tänker till vad som ändras mellan varje rad och kolumn i en av trianglarna. Uppgiften på nästa sida har dock en enklare lösning om du tänker på vilka brickor som tas med och hur de vänds.

Funktionella krav

Skriv ett program som avgör om en viss fil innehåller brickor som räcker till minst en komplett uppsättning av ett visst set. Användaren matar in högsta siffran på det set som ska testas och filnamn. Om inmatad siffra inte kan tolkas som ett heltal, eller om filen inte går att öppna ska programmet omedelbart avsluta med felmeddelande enligt körexempel. För att lagra brickor på fil används följande filformat. En bricka lagras som två heltal separerade med ett vertikalstreck, $x|y$ eller t.ex. 15|7. Olika brickor är separerade med minst ett blankt tecken.

Ickefunktionella krav

- Om du inte använder en `std::tuple` eller ett `std::pair` (rekommenderas) ska en Dominobricka representeras med en därför lämplig klass.
- Utskrifterna ska matcha körexemplen till fullo.

Tips och begränsningar

- Du kan utgå från att dominofiler enbart innehåller korrekta brickor (i något set) och blanka tecken.
- Du får använda valfri STL-container för att lagra instanser av dina brickor, men det finns förstås en container som ger mindre kod i övrigt.
- Tänk på att 21|7 och 7|21 räknas som samma bricka vänd åt olika håll.

Körexempel (användarinmatning kursivt)

```
$ ./a.out
Mata in högsta siffran i det set du vill ha: jag älskar C++
FEL: 'jag älskar C++' är inte ett heltal.

$ ./a.out
Mata in högsta siffran i det set du vill ha: 21
Mata in namnet på filen med brickor: trimino.txt
FEL: 'trimino.txt' kunde inte öppnas.

$ ./a.out
Mata in högsta siffran i det set du vill ha: 7
Mata in namnet på filen med brickor: double-six.txt
Brickorna räcker inte till en hel uppsättning "double-7".

$ ./a.out
Mata in högsta siffran i det set du vill ha: 9
Mata in namnet på filen med brickor: 1000_random.txt
Det finns minst en uppsättning för "double-9" i filen '1000_random.txt'.

$ ./a.out
Mata in högsta siffran i det set du vill ha: 10
Mata in namnet på filen med brickor: 1000_random.txt
Brickorna räcker inte till en hel uppsättning "double-10".
```

Uppgift 3 Kö (given_files/test_queue.cc, given_files/queue.h)

Inledning

I uppgift 1 finns behov av en kö. En kö börjar tom. Nya värden läggs alltid till sist, ett värde i taget, men till skillnad från en stack plockas värden ut framifrån. Värden kommer alltså ut i samma ordning som de stoppades in. Detta brukar även kallas FIFO (First In, First Out) eller FCFS (First Come, First Served).

Funktionella krav

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

Ickefunktionella krav

1. Din kö ska byggas upp av köelement där vardera köelement pekar på nästa köelement.
2. Din kö ska ha korrekt minneshantering.
3. Din kö ska använda god abstraktion och inkapsling.
4. Operationerna på kön ska heta *push* för insättning, *pop* för borttagning och *empty* för att kontrollera om kön 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

- Det är lämpligt att ordna kön på sådant sätt att insättning kan ske enkelt via en slutpekare, och borttagning kan ske lika enkelt via en startpekare.
- Filen `given_files/queue.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)

```
empty queue
testing 1 2 3
Ctrl-D
testing 1 2 3 empty queue
empty queue
```