TDP004 - Tentamen

2015-01-14

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.

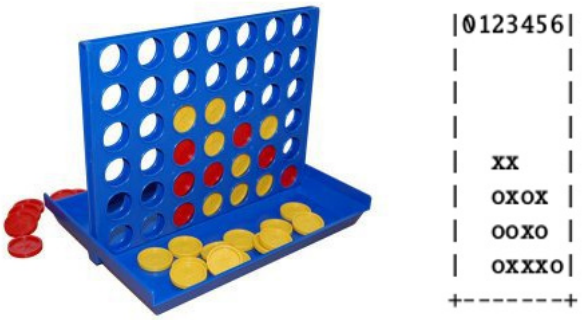
Avslutning

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 Fyra i rad (tillgodo om du löst uppgift 1 på förtentan)

Inledning

Fyra i rad är ett enkelt spel för två spelare. Spelet består av ett stående bräde med 7 rader och 7 kolumner. På sin tur släpper spelaren ned en bricka med sin färg i valfri kolumn. Brickan faller ned så långt den kan. Den som först får 4 brickor i rad horisontellt, vertikalt eller diagonalt vinner.



Figur 1: Fyra i rad på riktigt och i ascii, gul (kryss) har vunnit

Funktionella krav

Kim har implementerat spelet men tyvärr råkat radera det färdiga programmet. Hen har till slut lyckats hitta en gammal ofärdig fil `given_files/four-in-a-row.cc`. Din uppgift är att återskapa programmet så att spelet blir helt funktionellt. En tydlig representation av brädet ska skrivas ut efter att varje spelare placerat sin bricka.

Ickefunktionella krav

- Spelbrädet ska representeras med 7 rader med 7 tecken på varje rad. Numreringen börjar på 0 och slutar på 6. Blanktecken används för tomma positioner och 'x' resp 'o' för spelarnas brickor.
- Spelbrädet ska nyttja inkapsling och `const` så långt det går.
- Implementationen av spelbrädet ska ligga på egna filer.

Tips och begränsningar

- Du får använda valfri STL-behållare för att representera spelbrädet inuti din klass.
- Körexemplet är förkortat. Endast inledande inmatning och avslutande utskrift är med.
- Inmatningen i exemplet ger ställningen i figur 1 när spelet är slut.

Körexempel (användarinmatning kursivt)

```
x chose column: 4 2 3 3 5 6 4 5 3 4 5 2 3 2 2

...

'x' wins!
```

Uppgift 2 Obfuscate

Inledning

Som bekant betyder *obfuscate* att fördunkla något, göra det mindre uppenbart, lite som kryptering.

Funktionella krav

Du ska skriva ett program som läser in ett meddelande som kan vara flera rader. Inmatningen avslutas med filslut (Ctrl-D). Meddelandet ska sedan skrivas till en fil i mindre uppenbar form (d.v.s. "krypterat") enligt följande metod:

1. Sätt X till 1.
2. Skriv X:te tecknet i klartexten till kryptot.
3. Slumpa fram ett tecken mellan blanksteg (' ') och tilde ('~') (det är mellan 32 och 126 i asciitabellen) och skriv det slumpade tecknet till kryptot.
4. Ta fram slumpade tecknets position i asciitabellen (du får fram den automatiskt om du bara använder teckenvariabeln som ett heltal). Lagra positionen i N.
5. Slumpa fram ytterligare N tecken mellan ' ' och '~' och skriv dem till kryptot.
6. Sätt X till X+1 och börja om från 2.

Ickefunktionella krav

- Filnamnet ska anges på kommandoraden.
- Alla upptänkliga fel i samband med kommandorad och filöppning ska detekteras och programmet omedelbart avsluta med relevant felmeddelande.
- Felmeddelanden ska skrivas till standard felström.
- Du ska använda `std::random_device` och `std::uniform_int_distribution` för att få fram slumptal.
- Du får högst lagra ett tecken av klartexten åt gången i minnet.
- Avkrypterade meddelanden ska matcha originalet exakt, ned till varje blanksteg och nyrad.

Tips och begränsningar

- Du kan använda `given_files/de-obfuscate` för att plocka fram klartexten ur ett krypto.

Körexempel (användarinmatning kursivt)

```
$ ./a.out hej-pa-dig.txt
Hej pa
dig!
$ ./given_files/de-obfuscate hej-pa-dig.txt
Hej pa
dig!
```

Uppgift 3 Stack (given_files/test_stack.cc, given_files/stack.h)

Inledning

En stack börjar tom. Nya värden läggs alltid till sist, ett värde i taget, men till skillnad från en kö plockas värden även ut bakifrån. Värden kommer alltså ut i omvänd ordning mot vad de stoppades in. Detta brukar även kallas LIFO (Last In, First Out). Eller mer sällsynt LCFS (Last Come, First Served).

Funktionella krav

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

Ickefunktionella krav

1. Din stack ska byggas upp av stackelement (barr) där vardera barr pekar på nästa barr.
2. Din stack ska ha korrekt minneshantering.
3. Din stack ska använda god abstraktion och inkapsling.
4. Operationerna på stacken ska heta *push* för insättning, *pop* för borttagning och *empty* för att kontrollera om stacken 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 stacken på sådant sätt att insättning kan ske på enkelt sätt utan att iterera genom hela stacken.
- Filen `given_files/stack.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 stack
testing 1 2 3
Ctrl-D
3 2 1 testing empty stack
empty stack
```

Uppgift 4 Inläsning av vektor

Inledning

Den här uppgiften handlar mer om att lösa ett problem för programmerare än att lösa ett problem för slutanvändaren. Du ska göra det lätt att läsa in och skriva ut en `std::vector<int>`. Detta ska gå att göra med formaterad inmatning (>>) i koden. Eftersom rätt utskrifter nu inte betyder att koden är lättanvänd finns ett enkelt givet huvudprogram `given_files/vector-io.cc`. När befintliga koden går att kompilera och köra korrekt så har du antagligen gjort rätt.

Funktionella krav

Vi bestämmer att en vektor startar med en öppnande hakparentes. Därpå följer noll eller flera heltal separerade med minst ett blanksteg. Vektorn avslutas med minst ett blanksteg före en stängande hakparentes. Vid utskrift av en vektor gäller samma regler, men varje heltal separeras med minst ett blanksteg och använder minst 5 teckenpositioner. Även tomma vektorer ska hanteras (se exempel).

Ickefunktionella krav

1. Du ska implementera inläsnings (>>) och utskriftsoperatorn (<<) för `std::vector<int>`.
2. Utskrifterna ska matcha körexemplen till fullo.

Tips och begränsningar

Du kan anta att användaren matar in antingen helt korrekta vektorer eller vektorer som inte ens startar med hakparentes. Här är manipulatorn `ws` bra för att hoppa över blanka tecken i inmatningen och `peek()` är bra för att titta på nästa tecken i inmatningen utan att läsa det.

Körexempel (kommandorad fetstilad, användarinmatning kursiv)

```
$ ./a.out
Enter a vector of integers:
  [ ]
[ ]
remain in buffer: ''
$ ./a.out
Enter a vector of integers:
[1 2 3 4 5 6 7 8 9 ] remaining data
[    1    2    3    4    5    6    7    8    9 ]
remain in buffer: ' remaining data'
$ ./a.out
Enter a vector of integers:
    [ 1 2 333 4 4 4 4 555555 6 6 ]
[    1    2  333    4    4    4    4 555555    6    6 ]
remain in buffer: ''
$ ./a.out
Enter a vector of integers:
  this is not a vector
[ ]
remain in buffer: 'this is not a vector'
```

Uppgift 5 Old MacDonald

Inledning

Old MacDonald had a farm börjar texten i en engelsk barnvisa. Varje vers ser ut på samma sätt, men det aktuella djuret och dess läte är utbytt.

Funktionella krav

Skriv den saknade klassen `Animal` och skriv därpå tre subklasser som representerar var sitt djur och dess läte. I `given_files/animal_sounds.txt` finns lite inspiration till djur och läten. Slutför sedan huvudprogrammet enligt de givna kommentarerna.

Ickefunktionella krav

1. Ingen av klasserna ska lagra någon datamedlem.
2. Du får inte modifiera funktionen `print_verse` på något sätt.

Körexempel

```
Old MacDonald had a farm, E-I-E-I-O,
And on that farm he had a cow, E-I-E-I-O,
With a moo moo here and a moo moo there
Here a moo, there a moo, everywhere a moo moo
Old MacDonald had a farm, E-I-E-I-O.

Old MacDonald had a farm, E-I-E-I-O,
And on that farm he had a cat, E-I-E-I-O,
With a miow miow here and a miow miow there
Here a miow, there a miow, everywhere a miow miow
Old MacDonald had a farm, E-I-E-I-O.

Old MacDonald had a farm, E-I-E-I-O,
And on that farm he had a dog, E-I-E-I-O,
With a woff woff here and a woff woff there
Here a woff, there a woff, everywhere a woff woff
Old MacDonald had a farm, E-I-E-I-O.
```