TDP004 - Tentamen

2018-08-28

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 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. 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. *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	3	5
4 h	4	5
4 h	3	4
2 h	2	4
5 timmar	2	3

Tabell 1: Betygsättning vid sluttentamen, 5 uppgifter ges

Bonustid och tillgodoräknanden

Bonus och tillgodoräknade uppgifter gäller endast vid första ordinarie tentamenstillfället och alltså ej idag.

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++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 *experimentiell* tillgång till referenssidorna på `http://www.cppreference.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.

Testning av kod

Några av uppgifterna kan ha testfall skrivna med catch. För att ni inte ska behöva kompilera om catch vid varje ändring av er egen kod finns det en given fil `given_files/test_main.cc`. Denna behöver endast kompileras en gång och sedan kan man länka ihop den med de testfall man vill kompilera just nu. Gör följande:

1. Kompilera (men länka inte) `test_main.cc` till en objektfil (skapar filen `test_main.o` i nuvarande katalog):
   `g++17 -c given_files/test_main.cc`
   Observera att detta steg ska endast behöva göras en gång.

2. Vid kompilering av din kod läggs objektfilen till (skapar då en `a.out`):
   `w++17 test_main.o myfile.cc`

Uppgift 1 – DNA

Du ska ta över världen, men för att lyckas med detta behöver du skapa en grupp genmodifierade hjälpredor. Du har tyvärr inte tid att manuellt testa alla möjliga DNA sekvenser för dina hjälpredor, men du har kommit fram till att ett par ersättningsregler som transformerar om befintliga DNA strängar kan komma till användning.

(a) Du sammarbetar med din kollega Professor Vim för att ta fram dessa ersättningsregler:

```
A -> AT (utläses "A byts mot AT")
T -> A
```

Användaren matar in hur många gånger som ersättningsreglerna ska tillämpas.

Du börjar med den befintliga DNA sekvensen `"A"` och för varje tecken som har en ersättningsregel ska du ersätta tecknet med dess ersättningssekvens.

Upprepa proceduren på den nya DNA sekvensen antalet gånger som användaren begärde. Varje sekvens ska skrivas ut i terminalen.

(b) Åh nej! Atom-man kunde stoppa dina planer för att de tidigare ersättningsreglerna inte var optimala. Professor Vim hade glömt en vital del i sekvensen. Du konsulterar istället med din gode kamrat Dr. Emacs och tillsammans kommer ni fram till att dessa ersättningsregler fungerar bättre:

```
A -> AT
T -> GC
C -> TA
```

Krav

- Problemet ska lösas med någon lämplig container.
- För högre betyg krävs det att lösningen klarar både (a) och (b) delen, men det räcker med att endast (b)-delen skickas in.
- Fullständig uppräkning är ej godkänt.

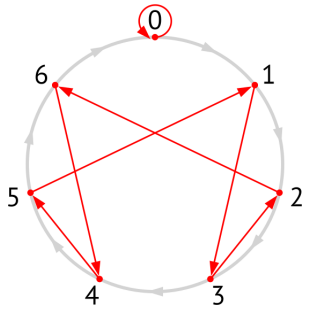
Körexempel (användarinmatning i fet stil)

(a)
```
Ange antalet iterationer: 4
A
AT
ATA
ATAAT
ATAATATA
```

(b)
```
Ange antalet iterationer: 5
A
AT
ATGC
ATGCGTA
ATGCGTAGGCAT
ATGCGTAGGCATGGTAATGC
```

Uppgift 2 – Delbarhet med 7

Det finns enkla metoder för att kontrollera huruvida ett tal är jämnt delbart med de första tre primtalen. Ett tal är jämnt delbart med 2 om det slutar på 0, 2, 4, 6 eller 8 och det är jämnt delbart med 5 om det slutar på 0 eller 5. Alla tal som är delbara med tre har egenskapen att summan av dess siffror är jämnt delbart med 3. Primtal större än 5 är inte lika enkla att kontrollera, men det finns ett lättare sätt att kontrollera snarare än att utföra divisionen. Givet grafen nedan kan vi kontrollera huruvida ett tal är delbart med 7 med följande algoritm:



1. Givet ett tal `n`, dela upp det i individuella siffror n_1, n_2, n_3 o.s.v. där n_1 är siffran längst till vänster (största positionen i talet).
2. Placera ett finger på 0 i grafen ovan.
3. Låt nuvarande siffra vara n_1.
4. Flytta ditt finger längs med med de gråa pilarna lika många gånger som värdet av den nuvarande siffran.
5. Om det finns fler siffror till höger om den nuvarande siffran, flytta ditt finger längs med en röd pil och börja om från steg 3 med nästa siffra. Om det inte finns några siffror kvar kommer ditt finger peka på värdet av `n % 7`.
6. `n` är delbart med 7 om du pekar på 0.

Som ett exempel, låt oss se om 243 är delbart med 7. Vi börjar på 0 och rör oss medurs 2 steg och landar på 2. Vi följer sedan den röda pilen till 6. Efter det fortsätter vi medurs 4 steg och hamnar på 3. Vi följer den röda pilen till 2 och tar sedan 3 steg medurs till 5. Det finns inga siffror kvar, därför vet vi att `243 % 7 == 5`, vilket inte är 0, så 243 är inte delbart med 7.

Funktionella krav

I filen `given_files/delbarhet.cc` finns ett testprogram som använder klassen `Mod7`. Din uppgift är att skapa en representation av grafen inuti klassen `Mod7` med hjälp av datatypen `Node` och sedan implementera ovanstående algoritm så att testfallen fungerar korrekt.

Uppgift 3 – Stapla klot

Inledning

Vi tänker oss N stycken genomskinliga glasrör ståendes på rad. Som ett spel kan man tänka sig att vi släpper ner klot (orbs) från ovan och låter dem landa i rören. Vi tänker oss att det är slumpmässigt vilket rör som just varje klot hamnar i. Vi antar även att rören är väldigt långa och smala och att det kan finnas maximalt 99 stycken rör.

Funktionella krav

Skriv ett program där användaren får mata in N, d.v.s. hur många rör vi har. Användaren skall därefter mata in hur många klot vi vill släppa ned (också maximalt 99 stycken). Programmet skall därefter slumpa hur kloten hamnar i rören och rita ut detta på skärmen. I den utritade "bilden" skall även rören vara numrerade från 1 upp till N. Ingen felhantering av indata krävs.

Ickefunktionella krav

- Du ska generera slumptal enligt C++11-standard.
- Din lösning ska använda minst en STL-behållare och minst en STL-algoritm.
- Din lösning ska ha minst en lämplig funktion utöver `main`.
- Utdata ska matcha körexemplen till formatet (inte exakt eftersom data är slumpat).

Körexempel (användarinmatning i fet stil)

```
Enter N: 5
Enter orb count: 13
          o
     o    o
     o    o  o
     o    o  o
  o  o    o  o
  1  2  3  4  5

Enter N: 4
Enter orb count: 7
        o
        o  o
  o  o  o  o
  1  2  3  4

Enter N: 13
Enter orb count: 7
                        o
  o  o              o     o     o     o
  1  2  3  4  5  6  7  8  9 10 11 12 13
```

Uppgift 4 – Ordfilter

Kopiera filen `given_files/ordfilter.cc` till din hemmapp. Lägg till din lösning i `ordfilter.cc`.

Ibland vill man kunna censurera sådant som skrivs eller sägs. I denna uppgift ska du skapa ett gränssnitt i form av klassen `Word_Filter`. `Word_Filter` är fullständigt abstrakt och har bara en funktion `approve` som tar in ett ord som en `std::string` och returnerar om ordet bör censureras eller ej. Det ska inte gå att skapa instanser av klassen.

Exakt hur vi avgör om ett ord är godkänt eller inte går att göra på många olika sätt. Du ska implementera tre olika varianeter i form av följande klasser:

- `Four_Letter_Filter` godkänner alla ord som inte är fyra tecken långa. T.ex. nekas "abra" medan "kadabra" godkänns. (Du behöver inte ta hänsyn till att vissa bokstäver (ääö mfl) kodas med två tecken, t.ex. är ordet "här" tre bokstäver men fyra teckenplatser långt och nekas.)
- `Wordlist_Filter` godkänner alla ord som inte förekommer i ett `std::set` med censurerade ord. Setet lagras i klassen och fylls på från en fil. Filnamnet skickas som parameter till klassens konstruktor. Setet har operationerna `insert` och `count` som kan vara användbara.
- `Allcaps_Filter` godkänner alla ord som har minst en gemen (nekar alla ord som enbart har versaler). T.ex. nekas ordet "HEJ" medan "Hej" godkänns.

Givet i `ordfilter.cc` finns klassen `Filter` som används av huvudprogrammet för att kunna kombinera flera olika `Word_Filter`. Tyvärr är den inte riktigt komplett, programmeraren har inte fått till vilken klass som ska hanteras i `add` och lagras i containern. Du ska korrigera detta i `Filter` och `main`.

Det är speciellt viktigt att få till en lösning som bygger på objektorienterade principer och ger enkla, oberoende och självständiga klasser. Minimera mängden datamedlemmar och if-satser.

Körexempel (användarinmatning i fet stil)

```
nu blev det dumknas i koden så att plötsligt ALLTING fungerar
nu **** det ******* i koden så att plötsligt ******* fungerar
```

Uppgift 5 – Blackjack

Blackjack är ett kortspel där målet är att komma så nära 21 som möjligt utan att bli "tjock" (få mer poäng än 21). I `given_files/blackjack.cc` finns det en enkel implementation av en förenkling av blackjack given där en spelare spelar mot dealern. Spelaren kommer dra kort tills dennes poäng överskrider 17. Dealern kommer att dra kort tills dennes poäng hamnar över spelarens poäng.

Det givna huvudprogrammet kommer att skriva ut vilka kort som spelaren respektive dealern drar och vem som vann i slutändan.

Din uppgift är att implementera en klass `Deck` som representerar en kortlek om 52 kort, indelad i 4 färger; spader, hjärter, ruter och klöver (engelska: spades, hearts, diamonds och clubs), där varje färg har 13 valörer associerade med sig; ess (ace), 2, 3, 4, 5, 6, 7, 8, 9, 10, knekt (jack), dam (queen) och kung (king). Det finns en `struct Card` given i koden som representerar ett kort.

I konstruktorn för `Deck` ska kortleken skapas med alla möjliga kort. Det finns en hel del funktionalitet som programmet förväntar sig av klassen, din uppgift är att ta reda på vad denna funktionalitet är och vad de ska göra. Tänk på vilken åtkomstnivå som alla medlemmar ska vara, och om det finns någon lämplig algoritm i standarbiblioteket som kan användas.

Funktionella krav

- Den givna koden får ej modifieras,
- Om möjligt ska standardbiblioteket användas,
- `const` måste användas där det är relevant,
- Det som är publikt i `Deck` är endast de medlemmar som används i huvudprogrammet, inget annat får förekomma i det publika gränssittet.

Körexempel

```
player hand:
2 of hearts
7 of clubs
player drew 5 of hearts
player drew ace of clubs
player drew 5 of diamonds
player stands with 20 points!
dealer hand:
queen of hearts
king of diamonds
dealer drew jack of spades
dealer is bust!
player won!
```