TDP004 - Dugga

2018-12-07

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 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	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 (dugga), 4 uppgifter ges

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++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.

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 - Yatzy

I spelet Yatzy ska spelaren kasta fem tärningar och välja vilka hen vill spara för att uppnå vissa kombinationer av tärningar. Några exempel på kombinationer är "ettor" där man endast vill få så många tärningar med en prick som möjligt, "par" då man vill ha två tärningar med samma värde och "liten stege" då man ska få en tärning av varje typ i intervallet 1 till 5.

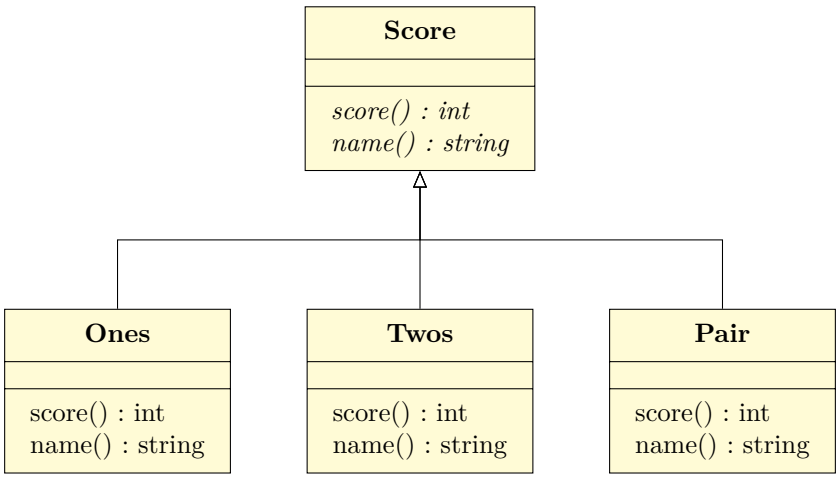
Funktionella krav

I denna uppgift ska du skapa en del av poängberäkningen i spelet Yatzy genom att skapa en klassheirarki där varje klass beräknar värdet hos en samling tärningar (som representeras av en `vector<int>`). De klasser du ska skapa ges av klassdiagrammet i figur 1. Kopiera filen `given_files/yatzy.cc` till din arbetskatalog och lägg till klasserna för att uppnå resultatet i kodruta 1.Klassernas score-funktioner ska göra följande:

- `Ones` och `Twos` ska returnera summan av alla ettor respektive tvåor som skickas in.
- `Pair` ska ge summan av det högsta paret.

Ickefunktionella krav

- Klassen `Score` ska vara en abstrakt klass
- Nyckelordet `const` ska användas på ett korrekt sätt där det passar
- Ingen av klasserna ska ha datamedlemmar



Figur 1: Klassdiagram för Yatzy
Parametertypen till medlemsfunktionen score är utelämnad av utrymmesmässiga själ

Kodruta 1: Utskrift vid korrekt implementation, obeservera att Pair alltid ger värdet av högsta paret

```
  Tärningar    1:or  2:or   Par
  1 2 1 3 4       2     2     2
  3 3 1 1 1       3     0     6
  5 5 6 6 3       0     0    12
```

Uppgift 3 - Optional

I c++17 finns det en ny typ med namn `std::optional` som *kan* innehålla ett värde av någon given typ. `std::optional` är bra att ha i gränssnitt, exempelvis som returvärden från en funktion som i vissa fall inte har något värde att returnera.

I denna uppgift ska du skapa en klass `Optional_Integer` som fungerar likt `std::optional`, men som endast kan innehålla ett heltal. Internt ska klassen lagra en pekare som pekar på ett dynamiskt allokerat värde, eller `nullptr` om klassen inte innehåller något värde.

Följande operationer ska stödjas:

Konstruktor En `Optional_Integer` ska kunna skapas från ett heltal, men även utan ett värde.

int & value() Returnerar en referens till det sparade värdet om det är satt men kastar undantag av en egendefinierad typ, `invalid_optional_access`, om inget är lagrat. Filen `given_files/optional.h` innehåller instruktioner på hur man skapar detta undantag.

Tilldelning Tilldelning ska fungera både från andra `Optional_Integer`-objekt och heltal.

void swap(Optional_Integer) Byter innehåll mellan två `Optional_Integer`.

bool has_value() Returnerar sant om vi har lagrat ett värde, annars falskt.

void reset() Tar bort det eventuellt lagrade värdet.

Filen `given_files/optional_test.cc` innehåller några catch-baserade testfall du kan använda för att testa din implementation.

Uppgift 4 - Taggar

Du har en fil, `given_files/names.txt`, där varje rad innehåller ett namn, ett kolon och flera mellanslagsseparerade taggar (ord). Din uppgift är att skapa ett program som låter användaren mata in ett antal taggar och sedan skriver ut alla namn som har samtliga taggar. Detta ska göras så långt som möjligt med hjälp av standardbibliotekets containrar och algoritmer enligt nedanstående algoritm:

1. Öppna filen för läsning
2. Skapa en map för att koppla en tag till alla namn som har den taggen (alltså rekommenderas typen `map<string, vector<string>>`) kallad `tags`. Observera att taggen alltså är nyckeln.
3. För varje rad i filen, läs in namnet och lägg till det i vectorn som hör samman med varje tag på raden.
4. Sortera alla namn-vektorer
5. Låt användaren mata in taggar, spara dem i en vector
6. Skapa en ny `vector<string>`, `common_names` och initiera den med namnen som hör samman med den första inmatade taggen
7. För alla inmatade taggar utom den första, skapa en ny `vector<string>` där du sparar de namn som förekommer både i `common_names` och bland namnen som hör till aktuell tag. Gör detta med hjälp av `set_intersection`. Skriv sedan över `common_names` med namnen i den nya vectorn.
8. Skriv ut namnen i `common_names`, ett namn per rad.

```
Mata in taggar: MEXICO SWEDEN
Gemensamma namn
SOFIA

Mata in taggar: SWEDEN NORWAY ICELAND FEMALE
Gemensamma namn
ANNA
EMMA
SARA
```