TDP004 - Tentamen

2018-04-06

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 – 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.

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 2 – Gruva

Du har blivit anställd av ett gruvföretag för att skriva ett program som ska hjälpa gruvarbetarna att hitta i gruvan. Det enda sättet att ta sig fram igenom gruvan är att åka med vagn i ett komplicerat rälssystem, där det finns en stor mängd korsningar. Din uppgift är att skriva ett program som låter gruvarbetarna att öva på vart de ska svänga i de olika korsningarna för att antingen hitta ut ur gruvan eller för att hitta till den nuvarande brytningsplatsen (där de bryter malmen). Självfallet kommer gruvan behöva utöka det här programmet i takt med att rälssystemet utvecklas och därför har företaget lagt stor vikt vid att programmet ska vara skalbart och använda sig av polymorfi.

Det finns tre typer av räls i systemet; ett rakt spår (klassen `Rail`), en korsning (klassen `Junction`) och en slutdestination (klassen `Destination`). Alla dessa klasser måste ha följande funktioner:

travel Funktionen som simulerar resan genom rälssystemet. Den ska endast ta en inström (`std::istream`) som argument, inget mer. Om rälsstycket kräver att användaren gör ett val så ska detta ske med den givna inströmmen. Denna funktion ska rekursivt anropa **travel** av nästa rälsstycke som vagnen ska åka till.

visit Den här funktionen kontrollerar huruvida vagnen har varit här innan. Om den har det ska ett lämpligt meddelande skrivas ut. Om den inte har det ska den markera att vagnen nu har varit här.

connect Den här funktionen används för att koppla ihop rälsstycken med varandra.

Exakt hur dessa funktioner fungerar varierar mellan de olika klasserna. Nedan följer en beskrivning av vardera klass.

Rail Denna klass representerar ett rakt rälsstycke, den ger inte användaren något val. `Rail` är även basklass för de andra två klasserna. Klassen ska även hålla reda på nästa rälsstycke.

- **travel** Börja med att anropa **visit**. Om rälsen inte leder till ett rälsstycke så ska meddelandet “Rälsen tog slut, du spårar ur!” skrivas ut på skärmen. Annars ska vagnen resa vidare till nästa rälsstycke.
- **visit** Om användaren redan har besökt det här rälsstycket ska meddelandet “Du känner igen dig...” skrivas ut, annars ska rälsstycket markeras som besökt.
- **connect** Denna funktion ska sätta vilket rälsstycke som är nästa (`nullptr` om rälsen tar slut).

Junction Denna klass representerar en korsning. Här har användaren två val; antingen ska vagnen fortsätta rakt fram, eller så ska den svänga.

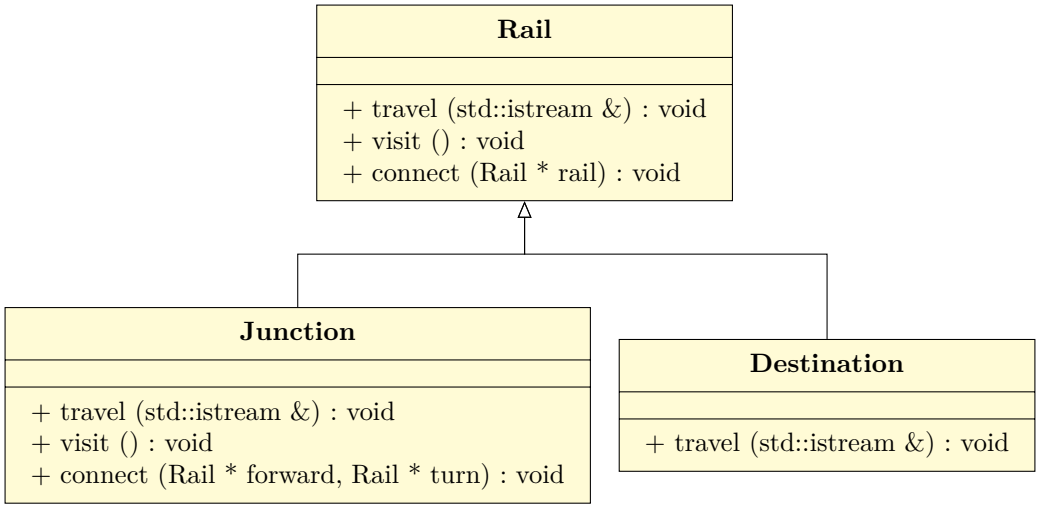
- **travel** Börja med att anropa **visit**. Följande meddelanden ska sedan skrivas ut “Vill du svänga (y/n)? ”. Om användaren matar in “n” så ska vagnen åka rakt fram, annars ska vagnen svänga. Det vill säga; programmet ska anropa `travel` på det rälsstycke som användaren vill åka till härnäst. Om nästa rälsstycke inte finns så ska beteendet bli detsamma som i `travel` i klassen `Rail`.
- **visit** Denna funktion ska först och främst skriva ut “Du känner igen den här korsningen” om vagnen har varit här tidigare, följt av vilket val användaren tog sist vagnen var här. Om användaren inte har varit här tidigare ska rälsstycket markeras som besökt.
- **connect** Denna funktion ska sätta vilka två rälsstycken som användaren kan välja mellan (`nullptr` om det det slut).

Destination Denna klass representerar en slutdestination som har ett namn.

travel Denna funktion ska skriva ut “Du kom fram till <namn>, bra jobbat!”. där <namn> ersätts med namnet på destinationen.

Fundera noga på exakt vad för datamedlemmar som bör finnas i varje klass och huruvida det finns gemensam funktionalitet som kan brytas ut i mindre hjälpfunktioner eller till basklassen.

Följande UML-diagram representerar hur kopplingen mellan klasserna ska vara. Observera dock att klassdiagrammet inte är fullständigt.



I `given_files` finns en fil `minecarts_test.cc` som ska fungera i samband med din kod. Du ska implementera `minecarts.h` och `minecarts.cc`.

Uppgift 3 – Filmer

Du ska skapa ett par klasser för att representera information om en film. En film har en titel, en längd samt ett antal roller. Titel och längd måste alltid anges när ett filmobjekt skapas och kan inte modifieras. Varje roll representeras av rollkaraktärens namn samt namnet på skådespelaren som spelar rollen. Dina klasser ska heta `Movie` och `Role`. Det finns ett givet huvudprogram på filen `given_files/movie_main.cc` som ska skriva ut information om filmen "Avatar" på formatet nedan när det fungerar.

```
Avatar [180 min]
Sam Worthington as Jake Sully
Zoe Saldana as Neytiri
Sigourney Weaver as Dr. Grace Augustine
```

Du måste dela upp din kod i en headerfil (`movie.h`) och en implementationsfil (`movie.cc`). Du lägger alltså deklarationer för båda klasserna i `movie.h` och definitioner i `movie.cc`. Rollerna ska internt i `Movie` sparas i en `std::vector` och skapas i medlemsfunktionen `add_role`. Det är såklart ett krav att lägga datamedlemmar och ansvar där det passar enligt god objektorienterad design. Du får inte göra några ändringar i den givna koden.

Uppgift 4 – Spel

Du och några vänner ska spela brädspel men har svårt att välja. Därför ska du skapa ett program som hjälper dig att slumpa ett spel.

Skapa en klass `Game` som representerar ett spel. Ett spel har ett namn, ett antal spelare (minimalt och maximalt antal) och en uppskattad speltid. `Game` ska även ha en medlemsfunktion `bool filter_players(int N)` som returnerar `true` om N är i intervallet för spelets rekommenderade antal spelare och `bool filter_time(int N)` som ger `true` om N ligger i intervallet speltid±20%.

Filen `given_files/games.cc` har en start på ett huvudprogram som du kan utgå ifrån. Ditt arbete är att bygga ut detta program så att det först frågar användaren om hur många spelare de är samt hur lång tid de har för att spela. Därefter ska programmet slumpa fram ett spel där både `filter_players` och `filter_time` ger `true` och skriva ut detta.

Nedan har du tre separata körexempel:

```
Hur många spelare: 3
Hur lång tid har ni (minuter): 45

Slumpat spel: Carcasonne
```

```
Hur många spelare: 1
Hur lång tid har ni (minuter): 60

Inga spel matchade :(
```

```
Hur många spelare: 6
Hur lång tid har ni (minuter): 35

Slumpat spel: Tsuro
```

Uppgift 5 – Olympier (Inte för TDDI14)

I mappen `given_files` finns en given indatafil, olympier.txt, med för- och efternamnen på de 1392 personer som har representerat Sverige i de Olympiska spelen från 1896 till och med år 2014.

Varje rad i filen innehåller ett förnamn och ett efternamn. Förnamnen består antingen av ett enkelt namn, som Anna, eller ett sammansatt namn med bindestreck, som Anna-Karin. Mellan förnamnet och efternamnet finns ett eller flera mellanrumstecken. Efternamnen kan se ut hur som helst (en eller flera delar, bindestreck) men ska helt ignoreras av programmet efter att indatafilen lästs.

Skriv ett program som kan läsa en fil med namn enligt ovan och skriva ut en lista med de vanligast förekommande förnamnen och antalet förekomster av varje namn. Utskriften ska se ut enligt exemplet nedan, där programmet fått skriva ut de 10 mest förekommande namnen.

```
 1. Erik     38
 2. Sven     25
 3. Gösta    24
 4. Carl     23
 5. Nils     21
 6. Gunnar   20
 7. Anna     19
 8. Per      18
 9. Johan    18
10. Anders   18
```

I exemplet ovan är det tre namn som är lika vanliga, Per, Johan och Anders (18). De borde egentligen hamna på samma plats (8), men sådant bortser vi från, liksom att namn som förekommer lika många gången kanske borde komma i bokstavsordning, etc.

Namnet på indatafilen samt antalet rader som ska skrivas ut anges som kommandoradsargument. Dessa ska så klart rimlighetskontrolleras.

Problemet ska lösas med någon väl vald standardbiblioteksbehållare.