TDIU20 – Tentaregler

Inloggning

Logga in i tentasystemet genom att välja session ”exam system” och logga in med ditt vanliga LiU-ID. Välj inte att ha denna session som standardsession. Verifiera att dina uppgifter stämmer och förbered din tentaplats. Som vanligt är det inte tillåtet att ha väskor eller jackor vid sin skrivplats och mobiltelefoner ska ligga avstängda i jacka eller väska. Ta fram ditt LiU-kort och invänta tentavakt för att få ett engångslösenord.

Hjälpmedel

Följande får tas med på tentan:

- En bok om c++. För boken gäller följande regler:
  - Kommentarer/noteringar som direkt rör text och exempel på sidan i fråga får finnas i sidmarginalen.
  - Egna sidflikar för att enkelt kunna hitta t.ex. de olika kapitlen är tillåtna.
  - Inga extra ark eller lappar, lösa eller fastsatta, får finnas.
  - Tomma sidor, insidan av pärmarna, försättsblad, etc., får inte innehålla programkod.
- Ett A4-ark med valfritt innehåll på vardera sidor (går ej att ersätta med två enkelsidiga ark).
- Penna för att anteckna under tentan. Ni kommer förses med blanka papper

Följande får INTE tas med:

- Elektroniska hjälpmedel såsom miniräknare och mobiltelefon

Utloggning

När du är nöjd med ditt betyg (som står i tentaklienten) kan du avsluta och logga ut som vanligt i menyn. Klicka sedan på ok följt av knappen avsluta tentan. Observera att du när du gjort detta inte kan logga in igen. Lämna inte din plats innan vanliga inloggningsskärmen syns.

Tentaregler

Tentan består av fyra uppgifter klassificierade som grundnivå eller avancerad. För godkänt betyg på tentan krävs lösning av en uppgift på grundnivå. För betyg 4 krävs dessutom lösning av en avancerad uppgift och för betyg 5 krävs en grunduppgift samt båda avancerade uppgifterna.
För att en uppgift ska anses godkänd krävs följande:

- att man noga följt alla instruktioner och krav ställda i uppgiften
- din kod följer god programmeringsstil (se labseriens rättningsguide)
- att klasser har ett tydligt ansvar och funktioner har en väl definierad uppgift
- att din kod har bra inkapsling och resurshantering

Bonus från labserien

Om du har bonus från labserien minskas antalet avancerade uppgifter som krävs med ett, dvs för betyg 4 krävs endast en grunduppgift och för betyg 5 krävs en av varje. Bonusen är endast giltig det år den erhölls.

Frågor om uppgifter

Frågor om tentan i stort eller uppgiftspecifika frågor ska ställas via tenta-klienten. Detta för att vi ska ha en historik av konversationen samt för att vi ska kunna ge samma hjälp till olika studenter.

Systemfrågor

Om du har systemfrågor som t.ex. problem med tentaklienten eller terminalen räcker du upp handen så kommer en assistent och hjälper till.

C++ referens

Det finns tillgång till valda delar av `cppreference.com`. Du måste starta webbläsaren via menyn för att komma åt sidan.

Alias för kompilering

Det finns tre alias att använda sig av för kompilering med c++17:
`g++17` Kompilering utan varningar.
`w++17` Rekommenderas!
`e++17` Kompilering med alla varningar som fel.

Uppgift 1 - Grundnivå

JSON är ett vanligt format för att serialisera (skicka som text) data. I denna uppgift antar vi att användaren ska skriva in information om ett antal studenter som anges på ett kolonseparerat format (FÖRNAMN:EFTERNAMN:LiU-ID:KURSREGISTRERINGAR). Ditt program ska läsa in studenter till EOF (Ctrl-D), spara all information om varje student som en klass `Student`, och sedan skriva ut informationen i JSON-format. Nedan följer ett exempel på tre studenter och utskriften som ska genereras från den inmatningen:
Indata (där samtliga namn och LiU-idn är helt slumpmässigt genererade):

```
Jonna:Curovac:JonCu573:9MGA31
Adam:Vilkancis:AdaVi732:TAOP52 LSMAC4
John:Nordin:JohNo286:
```

Förväntat utdata är då enligt nedan. Hela programmets utmatning är inom hakparenteser och varje student tar en rad utskrift. En student innesluts med klammerparenteser och fälten beskrivs med ord. Alla strängar är inom citationstecken. Studenter separeras med komma.

```
[{"Name": "Jonna", "Surname":"Curovac", "ID": "JonCu573", "Courses":
["9MGA31"]},
{"Name": "Adam", "Surname": "Vilkancis", "ID": "AdaVi732", "Courses":
["TAOP52", "LSMAC4"]},
{"Name": "John", "Surname": "Nordin", "ID": "JohNo286", "Courses": []}]
```

Krav

1. Informationen om en student ska sparas i en klass med namn `Student`. Självklart är det viktigt med god inkapsling.
2. Inläsning av en student på indataformatet ska ske med hjälp av överlagring av `operator>>`.
3. Utskrift av en student på JSON-format ska ske genom överlagring av `operator<<`.

På filen `given_files/STUDENTS.TXT` finns en lista av studenter enligt indataformatet du kan testa ditt program med.

Uppgift 2 - Grundnivå

En akademisk rapport har alltid en titel, en eller flera författare och ett antal kapitel. Ett kapitel har alltid en rubrik, en inledning och eventuellt ett antal underkapitel (som i sin tur kan representeras som ett kapitel). Din uppgift är att skapa två klasser, `Report` och `Chapter`, som kan användas för att representera detta. Följande krav ska uppfyllas av dina klasser:

Report

- Har följande *publika* medlemsfunktioner:

  Konstruktor Man måste ange rapportens titel när den skapas. Titeln ska aldrig gå att ändra.

  authors Ger en *modifierbar* `vector` av strängar som innehåller rapportens författare.

  new_chapter Ska skapa ett nytt kapitel. Tar emot en sträng (rubriken på det nya kapitlet) och returnerar en referens till kapitlet så att det kan modifieras av användaren. Kapitlet måste även lagras internt i klassen. Detta är det enda sättet att skapa ett nytt kapitel.

  print Tar en utström som enda parameter och skriver ut hela rapporten på den strömmen. Först skrivs titeln ut på egen rad, därefter författarna (separerade med komma) följt av kapitlen. Varje kapitel separeras med en tom rad (två newline-tecken).

- Ska inte gå att kopiera.

Chapter

- Går endast att skapa via `Report::new_chapter`
- Går inte att kopiera.
- Lagrar internt sin titel och inledning som strängar samt har en vector av underkapitel.
- Har följande *publika* medlemsfunktioner:

  text Ger en modifierbar sträng som motsvarar inledningen.

  add_subchapter Tar, likt `new_chapter` i Report, en sträng som motsvarar underkapitlets rubrik och skapar ett nytt underkapitel som både sparas internt och returneras till användaren som referens.

  print Tar en utström som enda parameter och skriver ut kapitlet på den strömmen. Först skrivs rubriken ut på en egen rad därefter inledningen och till slut underkapitlen separerade med tomma rader.

Filen `given_files/article.cc` innehåller ett testprogram som ska fungera om dina klasser uppfyller målen ovan.

Uppgift 3 - Avancerad

Ibland kan det vara intressant att hålla koll på vad som har hänt med ett objekt över dess livstid. I denna uppgift ska du implementera en programkomponent, `Lifetime_Logger`, som ska kunna användas för att göra just detta.

Tanken är att om jag som programmerare vill veta vad som händer med just min klass kan jag lägga till en datamedlem av typen `Lifetime_Logger` som sedan gör jobbet. Ett `Lifetime_Logger`-objekt har internt en lista (`vector<string>`) över tidpunkter då något speciellt hände med objektet (exempelvis att det skapas eller kopieras). Varje gång ett objekt kopieras eller flyttas ifrån ska detta loggas, både i originalet och i kopian. Filen `given_files/logger.cc` innehåller en given fil som dels har en funktion som ger nuvarande tid som en sträng dels ett testprogram som skapar och hanterar ett antal objekt av typen `Lifetime_Logger`.

Utöver speciella medlemsfunktioner får `Lifetime_Logger` endast ha en publik medlemsfunktion `print_log`. Denna funktion tar emot en utskriftsström och skriver ut samtliga logmeddelanden på strömmen, ett meddelande per rad.

Uppgift 4 - Avancerad

I en välkänd amerikansk restaurangkedja kan man köpa en så kallad pankaksstack. Det är alltså en hög med pannkakor staplade på varandra. Man kan även beställa tillbehör på varje pannkaka, exempelvis kan man beställa en bananpannkaka, på den ha en pannkaka med strössel och chokladsirap och ovanpå den ha en pannkaka med lönnsirap. Priset på hela beställningen beror både på antalet pannkakor och vilka tillbehör de har. Din uppgift är att skapa en klassheirarki för att representera pannkakorna och tillbehören.

Klassen `Pancake` representerar en pannkaka med tillbehör. Den har en medlemsfunktion `add` som tar emot ett tillbehör (beskrivs nedan) som argument och sparar det internt. Medlemsfunktionen `price` returnerar 0.75 (priset för en ensam pannkaka) plus summan av tillbehörens kostnader. Funktionen `eat` skriver ut ”Mmm, a pancake with: ” och anropar sedan `eat` för respektive tillbehör. Om pannkakan inte har några tillbehör ska `eat` istället skriva ut ”Hmm, a plain pancake.”

Klassen `Condiment` är en abstrakt basklass för att representera ett tillbehör. Varje tillbehör har ett pris (double) och en funktion för att hämta detta. Dessutom finns en medlemsfunktion `eat` som skriver ut en beskrivning av tillbehöret. Tre tillbehör (subklasser till `Condiment`) ska skapas, `Banana`, `ChokoBits` och `Syrup`. Egenskaperna hos Banana och ChokoBits hittar du i tabell 1. `Syrup` ska förutom pris lagra vilken typ av sirap det är och det anges vid konstruktion. Priset för Syrup är alltid $0.20 och `eat` funktionen ska inkludera typen, exempelvis kan den skriva ut strängen ”maple syrup”.

Klass	Pris	Utskrift från eat
`Banana`	$0.30	banana slices
`ChokoBits`	$0.45	chocolate bits!

Tabell 1: Egenskaper hos några tillbehör

Den givna filen `given_files/pancakes.cc` har ett skelett som du ska utöka enligt kommentarer i filen och informationen ovan.

Tillbehören ovan är minimikrav, men saknar du ditt eget favorittillbehör (vart är sylten?) får du gärna lägga till fler klasser för att representera dem.