TDIU20 – Tentaregler

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 har fyra praktiska uppgifter och en teoretisk uppgift. Två praktiska uppgifter är på grundnivå. Där visar du att du uppnår kraven för godkänt. De två praktiska uppgifterna och den teoretiska uppgiften är på djupare nivå. I dessa kan du visa att du nått upp till kraven för ett högre betyg. I den teoretiska uppgiften visar du att du kan använda rätt terminologi och förstått syften som behöver förklaras i text. Bedömning av den teoretiska uppgiften sker efter tentans slut. Testa dig själv genom att resonera kring instuderingsfrågorna.

För betyg 3 på tentan krävs lösning av en uppgift på grundnivå. För betyg 4 krävs dessutom lösning av en högre betygsuppgift och för betyg 5 krävs grunduppgift och två högre betygsuppgifter.
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 högre betyguppgifter 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 skrivborsikonen "Web access" för att komma åt sidan.

Alias för kompilering

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

Uppgift 1 - Grundnivå

Den nya spelstudion Oden söker nya medarbetare. Till ansökan ska lämnas ett arbetsprov enligt nedan.

I ett nytt spel ska spelaren skaffa resurser för att bygga sin medeltida flotta. Till detta behövs en skog och i skogen finns träd. Träd har ett trädslag och en ålder och består av många löv, grenar och kvistar. Träd har dessutom en höjd, en omkrets och ett djupt rotsystem.

För att beräkna mängden virke som erhålls när ett träd skördas ska träds volym kunna beräknas. Volymberäkningen går till så att volymen för trädets samtliga grenar summeras (kvistar och löv räknas inte in). Volymen för en gren approximeras med volymen av en cylinder enligt formeln $3.14 * length * radius^2$. Trädstammen kommer vara en gren i trädet så inget speciellt behövs för att få med den.

Skriv en klass som representerar ett träd av en viss art med alla dess grenar i en `std::vector`. Du behöver endast ha med medlemmar som krävs för att utföra volymberäkningen för hela trädet och skriva ut dess art och volym.

- Vi bedömer hur väl din lösning fungerar med en omodifierad version av `uppgift1.cc`.
- Vi bedömer din lösning utefter hur få kompileringsvarningar den får med `w++17`.
- Skriv lösningen med filuppdelning. Klasser som hör intimt samman får ligga i samma fil.
- Sammansatta datatyper ska parameteröverföras utan kopiering och utan risk för oavsiktlig ändring.
- Medlemsfunktioner ska vara användbara på konstanta objekt i så stor utsträckning som möjligt.
- Dina klasser ska vara väl inkapslade och följa god konvention. Det finns inget behov att använda "set"-funktioner eller vänfunktioner.

Körexempel 1

```
Trädet har 6 grenar och är av arten ask.
Volymen är 604.522 kubikmeter.
```

Uppgift 2 - Grundnivå

Din bästa kompis som pluggar kemi är frustrerad över hur besvärligt det är att räkna på koncentrationen när en lösning blandas med vatten. Du antar utmaningen och erbjuder dig att skriva ett program som utför beräkningarna om din kompis förklarar hur beräkningen ska utföras.

Skriv en klass `Water_Mixture` som representerar en blandning av vatten och en annan vattenlöslig vätska. En blandning består av en viss volym rent vatten och en viss volym ren annan vätska. Det ska gå att beräkna en blandnings:

- koncentrationen av vatten i procent.
- koncentration annan vätska i procent.
- totala volym vätska.

Det ska dessutom gå att plussa ihop två blandningar med den vanliga plusoperatorn. Resultatet ska bli en ny blandning. Vi antar (utan att hålla reda på det) att båda blandningarna består av samma vätskor. Det ska även gå att späda ut en blandning med mer vatten genom att addera ett heltal (volym vatten) till blandningen. Resultatet blir en ny blandning. Notera att vi tvärt emot intuitionen förväntar oss att originalblandningarna finns kvar opåverkade när blandningar hälls ihop eller späds (med plusoperatorn).

- Vi bedömer hur väl din lösning fungerar med en omodifierad version av `uppgift2.cc`.
- Vi bedömer din lösning utefter hur få kompileringsvarningar den får med `w++17`.
- Skriv lösningen med filuppdelning. Klasser som hör intimt samman får ligga i samma fil.
- Sammansatta datatyper ska parameteröverföras utan kopiering och utan risk för oavsiktlig ändring.
- Medlemsfunktioner ska vara användbara på konstanta objekt i så stor utsträckning som möjligt.
- Dina klasser ska vara väl inkapslade och följa god konvention. Det finns inget behov att använda "set"-funktioner eller vänfunktioner.

Körexempel 1

```
Lösningen loka är totalt 33 cl med 100% vatten och 0% etanol.
Lösningen absolut är totalt 200 cl med 5% vatten och 95% etanol.
Lösningen handsprit är totalt 253 cl med 24.9012% vatten och 75.0988% etanol.
```

Uppgift 3 - Påbyggnad för högre betyg

Den här uppgiften löser du när du har godkänt på en uppgift på grundnivå för att höja betyget.

Bra jobbat! Du kom vidare till intervju på Oden Games. Nu vill teknikchefen se hur du presterar i en pressad situation och ger dig följande uppgift.

Flottan spelaren byggt av sina träd består av skepp. Ett skepp är byggt av trä, har ett antal besättningsmedlemmar, en segelyta och ett deplacement (vikten av undanträngda vattnet utan last). En Fregatt är ett skepp som dessutom har ett antal kanoner. Ett Transportskepp har en lastkapacitet för värdefull last och ett passagerarantal.

I spelet används transportskeppen för att forsla hem så mycket värdefull last och passagerare som möjligt. Transportskeppen hotas av motspelarens fregatter och skyddas av de egna fregatterna. Alla skepp kan ta skada av en avlossad bredsida från en fregatt. En bredsida är alltid hälften av antalet kanoner på fregatten och noll för en transport. Resultatet beror av antalet avlossade skott vid en bredsida:

- En femtedel av skotten missar alltid målet oavsett typ av skepp som utgör mål.
- Är målet en transport fördelas övriga skott så att hälften skrämmer per skott en passagerare överbord, en fjärdedel skadar per skott 5% av lasten och varje skott efter det förstör 10% segelyta.
- Är målet en fregatt fördelas övriga skott så att ett skott sätter en besättningsmedlem ur spel, hälften minskar segelytan med per skott 5% och varje skott efter det sätter en kanon ur spel.

Varje skepp har även en funktion för att skriva ut sin status (antal oskadade kanoner, besättningsmedlemmar, segelyta, passagerare och last) så som är rimligt för respektive skeppstyp.

Beräkningarna för skada och andra detaljer du behöver finns förberedda i den givna filen `uppgift3.cc` där du lägger till all din kod. Det ingår i uppgiften att förstå och anpassa sig till den givna koden.

Skapa klasser för att representera Fregatter och Transportskepp. Vi förväntar oss att du använder alla lämpliga tekniker för att undvika koduppprepning. Du ser till att varken slicing eller minnesläckor förekommer. Med hänsyn till den pressade situationen behöver användningen av `const` inte vara perfekt. Du behöver enbart ha medlemmar för saker som behövs för att lösa uppgiften.

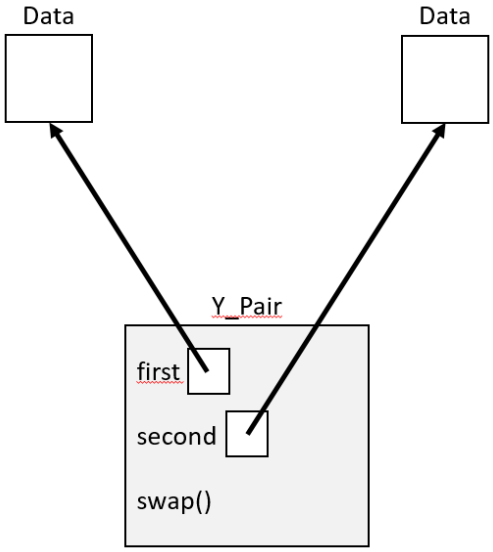
Körexempel 1

```
Transport with 15 crew, 1018.1 sq m sails, 65.0 ton cargo and 58 passengers.
Transport with 30 crew, 1303.3 sq m sails, 82.0 ton cargo and 43 passengers.
Frigate with 96 crew, 2301.4 sq m sails and 3 cannons.
Frigate with 96 crew, 2422.5 sq m sails and 4 cannons.
Frigate with 97 crew, 2565.0 sq m sails and 4 cannons.
Frigate with 97 crew, 2565.0 sq m sails and 7 cannons.
Frigate with 54 crew, 958.0 sq m sails and 0 cannons.
```

Uppgift 4 - Påbyggnad för högre betyg

Den här uppgiften löser du när du har godkänt på en uppgift på grundnivå för att höja betyget.

Din näst bästa kompis läser en kurs i programmering. Där har examinatorn hittat på en speciell datastruktur `Y_Pair` för att representera ett par värden av typen `Data`. Den speciella egenskapen med datastrukturen är att dess data ska kunna byta plats inom datastrukturen så effektivt som möjligt. Även data som inte kan kopieras ska kunna byta plats.



- Figuren beskriver hur datastrukturen ska organiseras i minnet.
- Funktionen `swap` beskrivs som att `first` och `second` ska byta plats.
- Det ska inte gå att skapa ett `Y_Pair` utan att ange både `first` och `second`. Både `first` och `second` ska vara dynamiskt allokerade och `Y_Pair` ska ta över ansvaret för minneshantering så fort objektet är skapat.
- Ett `Y_Pair` ska gå att flytta till ett annat med `std::move` men det ska inte gå att kopiera. Alla försök till kopiering ska ge kompileringfelet att funktionen för detta är antingen borttagen eller privat i klassen.

Visa hur datastrukturen implementeras. Din kod ska skrivas i sin helhet i filen `y_pair.h` och det givna huvudprogrammet ska fungera utan ändringar. Inga minnesläckor eller minnesfel ska förekomma.

Körexempel 1

```
Thelma and Louise
Louise and Thelma
Butch and Sundance
```

Uppgift 5 - Teoretisk påbyggnad för högre betyg

Den här uppgiften löser du när du har godkänt på en uppgift på grundnivå. Svaret skriver du i ren text i filen `uppgift5.txt`. Du vet hur du skriver bra kod - men kan du förklara vad du får ut av det? Du bedöms efter hur väl du använder korrekt terminologi och hur väl du motiverar dina argument.

Två lyckliga nördar Ellis och Aria ska programmera en telefonbok. Båda är överens om att telefonboken ska vara en |std::vector| med kontakter och att en kontakt är ett aggregat med namn och telefonnummer. De är överens om att det ska finnas funktioner för att lägga till kontakter, ta bort kontakter och söka ut en delmängd kontakter som matchar ett sökord:

```
struct Contact {
  std::string name;
  std::string phone;
};
std::vector<Contact> phone_book;

vector<Contact> search(std::string word);
void add_contact(Contact c);
void remove_contact(std::string name);
```

Men sedan börjar åsikterna gå isär. Ellis är praktiskt inriktad och menar att programmet alltid kommer ha exakt en telefonbok som hanteras och det därför blir bra med en global vektor. Då kan funktionerna bekvämt och effektivt komma åt vektorn utan extra kod. Det räcker just nu och jobbigare behöver det inte vara.

Aria som är mer visionär ser en framtid där telefonboken finns som molntjänst med en telefonbok per användare och att man dessutom vill hålla reda på adresser. Då tror Aria att det behövs mer än en sträng för att representera ett telefonnumer, med tanke på internationella användare med flera telefoner och landsprefix. Aria anser dessutom att koden ska licensieras till företag vill göra egna anpassade telefonboksappar. Aria förespråkar därför att göra telefonboken och dess funktioner som klasser trots att de just nu bara behöver en telefonbok och ett enkelt aggregat för en kontakt.

Ellis anser Aria tänker för mycket och att en så avancerad lösning bara blir extra många fel att reda ut innan allt kompilerar. Aria menar att kompileringsfel så tidigt som möjligt är bra.

Diskutera fördelar och nackdelar med en objektorienterad inkapslad lösning. Kommer de att vinna något på att göra mer arbete på klasser med inkapsling? Blir det mycket extra jobb? Vad blir egentligen skillnaden på att lägga kodexemplet ovan globalt (Ellis tanke) jämfört med att ha den i en klass `Phone_Book` (Arias idé)? Borde `Contact` vara en inkapslad klass? Hur kan Aria ens mena att kompileringsfel är bra? Vilka tekniker tycker du att de ska använda i sin lösning?

Använd begreppen *datamedlem*, *medlemsfunktion*, *konstruktor*, *instans*, *objekt* och *klass* i diskussionen. Motivera effekter på användbarhet i huvudprogram, felhantering, koduppprepning, ändringsbarhet och läsbarhet med dina förslag.