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

Du ska skapa ett par klasser för att representera och spara information om ett rymdskepp.

Skeppet har ett namn, en längd (i meter) och besättningsmedlemmar. När skeppet skapas skall alltid namnet på skeppet och dess längd anges. Besättningsmedlemmarna läggs till efter objektet skapats. En besättningsmedlem har ett namn och en rang, som båda anges när den skapas.
Det finns ett givet huvudprogram på filen `uppgift1.cc` som ska skriva ut information om skeppet ”Enterprise” när det fungerar.

Du måste dela upp din kod i en headerfil (`spaceship.h`) och en implementationsfil (`spaceship.cc`). Du lägger alltså deklarationer för båda klasserna i `spaceship.h` och definitioner i `spaceship.cc`. Besättningen ska internt i Spaceship sparas i en `std::vector` och läggs till med medlemsfunktionen `add_crew_member`. 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.

Körexempel:

```
name: Enterprise E
length: 685
crew:
Riker - First Officer
Picard - Captain
Wessley - Intern
```

Uppgift 2 - Grundnivå

Du ska skapa en klass för att hantera handelsgods.

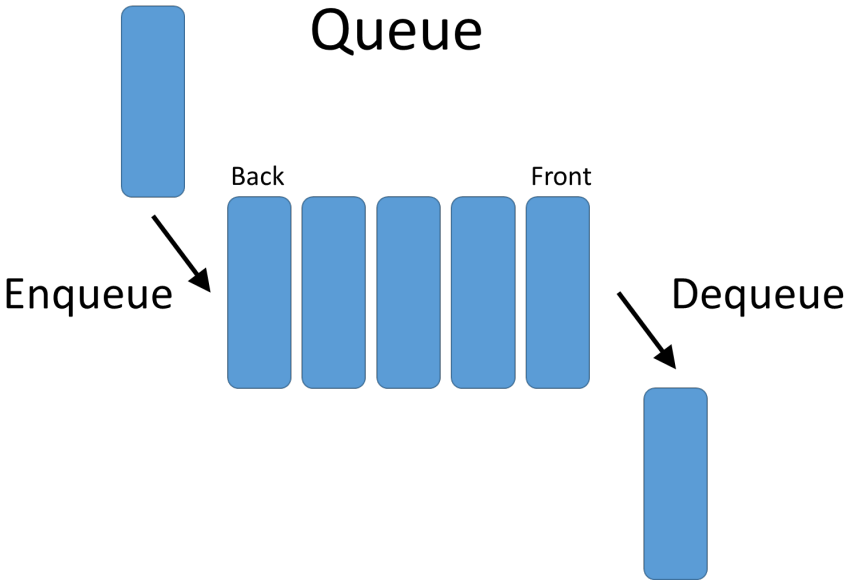
Ett handelsgods (`Cargo`) skapas alltid med namnet på godset och värdet av godset. Godset kan sedan skrivas till godtycklig ström av typen `std::ostream`. Godset kan också läggas ihop med ett annat gods med plus-operatorn. Om ett gods slås ihop med ett annat gods skall de ursprungliga objekten förbli oförändrade och ett nytt objekt skall skapas. Det nya objektet skall också vara av typen `Cargo` och objektets namn skall vara en sammanslagning av de ursprungliga objektens namn och dess värde skall vara summan av de ursprungliga objektens värden (se körexemplet)

Du skall skapa filerna `cargo.h` och `cargo.cc`. Din lösning skall sedan läggas till i dessa filer. Du skall fördela koden mellan filerna enligt god programmeringssed. Det finns given kod som kommer att använda klassen du skapar i en given fil (`given_files/uppgift2.cc`). Koden i denna fil får inte förändras och din klass skall fungera med den givna koden och enligt körexemplet.

Körexempel:

```
Beans 5
Latinum 10000
Tribbles -5000
BeansLatinum 10005
BeansLatinumTribbles 5005
```

Uppgift 3 - Avancerad nivå



Ombord varje skepp finns ett datorsystem. Detta datorsystem utför uppgifter enligt ett kösystem. Detta kösystem är skrivet i C++ och finns i filerna `queue.h` och `queue.cc`. Kön är en datorstruktur som består av `Tasks`. Varje task innehåller en sträng som beskriver vad uppgiften är och en perkare till nästa task. Kön är av typen FIFO (First In First Out). Det innebär i det här fallet att varje uppgift läggs till sist i kedjan av länkar och att den första länken i kedjan är den som först kommer lämna listan. Man kan tänka på det som en stack som fylls på underifrån.

Ditt jobb är att utöka klassen `Queue` så att den kan hantera kopiering, flytt och borttagning på korrekt sätt. Med andra ord behöver du lägga till en `Kopieringskonstruktor`, `Kopieringstilldelningsoperator`, `Flyttkonstruktor`, `Flytttilldelningsoperator` och en `Destruktor`. Du behöver göra detta enligt god objektorienterad praxis och med rätt uppdelning mellan filerna. När du är klar behöver du lägga till kod i `uppgift3.cc` för att visa att dina speciella medlemsfunktioner går att köra utan några problem. Den givna koden för kön får inte ändras och din kod bör fungera så effektivt och felsäkert som möjligt. Flytt och kopiering av kön ska också vara djup så att två köer inte hänvisar till samma tasks efter kopiering eller flytt.

Du kan köra din kod med `valgrind ./a.out` efter du kompilerat för att se att den fungerar utan minnesfel. Tänk på att inga minnesfel kommer uppstå om koden i huvudprogrammet inte testar den delen av klassen. Det är därför viktigt att du modifierar huvudprogrammet så att alla speciella medlemsfunktioner körs.

Uppgift 4 - Avancerad nivå

Federationens flotta innehåller många olika typer av skepp. De kan exempelvis vara fraktskepp, forskningsskepp eller krigsskepp. Din uppgift är att skapa en polymorfisk klasseirarki som representerar dessa olika skepp.

Skapa en abstrakt basklass, `Ship`, för att representera ett generellt skepp. Skapa sedan två konkreta (icke abstrakta) klasser som representerar fraktskepp (`CargoShip`) och krigsskepp(`WarShip`). Dessa klasser skall alla skrivas i filerna `ship.h` och `ship.cc`. I den givna filen, `uppgift4.cc`, används sedan dessa klasser för att skapa ett antal skepp och räkna ut den totala styrkan av federationens flotta samt skriva ut skeppen som finns i flottan. Det blir dock lite fel i `uppgift4.cc` vilket leder till slicing, och för att programmet skall fungera korrekt behöver du lösa det problemet.

Krigsskepp skapas med parametrarna namn(sträng), vikt(heltal) och vapen(heltal). Fraktskepp skapas med parametrarna namn(sträng), vikt(heltal) och lastutrymme(heltal). Alla skepps individuella styrka beräknas med medlemsfunktionen `get_power`. För ett krigsskepp beräknas styrkan med formeln `weight * guns`. För ett fraktskepp beräknas styrkan med formeln `weight + cargo_capacity`. Alla skepp kan också konverteras till en sträng för utskrift med funktionen `print_string()` (se körexemplet).

Din lösning skall inte ha några minnesläckor. Du kan köra programmet med `valgrind ./a.out` efter du kompilerat för att se om du läcker något minne. Tänk på att du med valgrind endast kan upptäcka minnesfel i kod som faktiskt körs.

Körexempel:

```
name: Sixth fleet
power: 227
name: Defiant, weight:8
name: Valiant, weight:8
name: Yoshiro, weight:40
name: Enterprise, weight:20
```