TDP004 - Tenta

2020-12-07

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 och 2. *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

Tid	**Lösta uppgifter**	**Betyg**
3 h $+B$	3	5
4 h $+B$	4	5
4 h $+B$	3	4
2 h $+B$	2	4
5 timmar	2	3

Tabell 2: Betygsättning vid sluttentamen, 5 uppgifter ges

Bonustid ($+B$)

All bonustid gäller endast under den första ordinarie tentamen i samband med kursen (januari). Varje moment i kursen som ger bonus ger 5 minuter extra tid för högre betyg på sluttentamen, upp till maximalt 45 minuter. Detta är markerat med $+B$ i tabellen där bonus räknas.

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

Logga in på datorn i labbsalen med ditt liu-id och lösenord. Detta är samma inloggningsuppgifter som du använder i Lisam.

Skrivbordsmiljön

När du kommit in i tentasystemet har du en normal skrivbordsmiljö (Mate-session i Ubuntu). 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 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. Tag hjälp av assistent i sal om det inte fungerar.

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.

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 logga ut ur datorn. Lämna inte datorn innan du ser att du är utloggad.

Uppgift 1 - Grundläggande klasser - Bokhylla - Tasha

Tasha är säkerhetschef på Enterprise. Förutom att ha ett något hett temperament så är ett utmärkande drag att hon gillar att läsa böcker. Hon har bokhyllor i sin hytt som är belamrade med böcker. Tyvärr har hon svårt att uppskatta hur mycket vikt varje hylla faktiskt klarar av att hålla och när hon skaffar en ny bok vet hon inte vilken hylla hon ska ställa den på. Därför vill hon att du ska skapar klasser för att lösa detta problem.

Du ska skapa klasserna som krävs för att huvudprogrammet ska fungera, huvudprogrammet hittar du i `given_files/uppgift1.cc`. Klasserna du behöver skapa är minst en som representerar bokhyllan och en som representerar en bok. Klassen som representerar en bok kan vara ett aggregat (aggregate class) om du vill, men objekt av denna klasstyp ska kunna skapas genom att användaren skickar in bokens namn och vikt.

Bokhyllan ska fungera som följer. Objekt av denna klasstyp skapas genom att användaren skickar med 1 parameter: hur mycket vikt hyllan klarar. Bokhyllan ska ha 2 publikt tillgängliga funktioner `put_book` och `to_string`.

- `put_book` ska ta emot en parameter av typen bok (Book) och saknar returvärde. Funktionen skall sedan lagra boken i bokhyllan om den klarar av att bära vikten av denna nya bok och alla tidigare böcker som lagrats. Om inte hyllan håller för att lagra denna nya bok ska den inte läggas till.
- `to_string` ska inte ta emot några parametrar och returnerar en strängrepresentation av objektet. Strängrepresentationen ska se ut som körexemplet.

Tips: Välj en lämpliga databehållare för att lagra böckerna i hyllplanet.

Tips: Som du ser i körexemplet kan en lättare bok passa på en hylla trots att tidigare böcker inte fått plats där. ”The Pickaxe” läggs exempelvis till efter C++ Primer i huvudprogrammet men får plats i hyllan.

```
/a.out
The Hobbit Learning Python The Pickaxe
```

Uppgift 2 - Polymorfi - Växande växter - Phlox

Phlox har fått i uppdrag att katalogisera och bestämma värdet av en skog över tid. Han har därför hyrt in dig för att skapa en polymorf klasstruktur för att representera träd.

Träd är basklassen (Tree) och ska kunna skapas genom att användaren skickar med longituden och latituden. Alla träd har koordinaterna och en höjd. Höjden för ett nytt träd börjar alltid på 0.

Följande Funktioner ska finnas för träd:

- man ska kunna hämta ut ett träds nuvarande värde med en funktion som heter `get_value`. Det finns dock inget gemensamt sätt att beräkna detta värde.
- träd skall också kunna omvandlas till en sträng med funktionen `get_info`. `get_info` ska för alla träd returnera en sträng som innehåller koordinaterna samt höjden för trädet, se exempelutskriften nedan.
- träd har också en funktion `grow` vars grundbeteende är att trädets höjd ökar med 1

Ek (Oak) är en klass som härleds från Träd som utökar Träds funktionalitet enligt följande:

- `get_info` lägger nu till att trädet är en ek (se körexemplet för hur strängen ska se ut för ekar)
- `get_value` beräknas för ekar genom: höjden * 5

Äppelträd (Apple_tree) är en klass som härleds från Träd och utökar funktionaliteten enligt följande:

- Sparar också antalet frukter som trädet bär. Ett nytt träd har alltid 0 frukter
- `get_info` lägger nu till att trädet är ett äppelträd (se körexemplet)
- `get_value` beräknas för äppelträd genom: (höjden * 3) + frukter
- `grow` ökar nu också antalet frukter som trädet bär med 1

Förutom att implementera dessa klasser ska du också skriva klart det givna huvudprogrammet.

Tips: Tänk på att inte duplicera kod i onödan

Tips: Tänk på att kontrollera eventuella minnesläckor

```
$ ./a.out
Oak tree
Coordinates - LON. 3, LAT. 4
Height: 5
Worth: 25

Apple tree
Coordinates - LON. 5, LAT. 7
Height: 5
Worth: 20
```

Uppgift 3 - Avkodning av meddelanden

Spock vill skapa ett program som kan avkoda meddelanden som han har hittat på en främmande planet. Han har listat ut hur avkodandet ska gå till men skulle vilja skapa ett program för detta för att effektivisera arbetet. Han har fått höra att standardbiblioteket i C++ är väldigt effektivt att använda och har även hört att du precis har lärt dig en massa om standardbiblioteket. Spock skulle därför vilja att du implementerar ett program som tar emot ett filnamn, avdkodar meddelandet och skriver sedan ut det avkodade meddelandet i terminalen. Stegen som Spock vill att du använder för att avkoda meddelandet är följande:

1. Läs in från en fil (filnamnet tar du in via kommandoraden) och spara innehållet i en lämplig behållare
2. Ta bort ord som inleds med `"#"`
3. Vänd ordningen på orden i behållaren, `C++ is the best => best the is C++`
4. Vänd varje ord bak och fram, `hej => jeh`
5. Skriv ut meddelandet med `cout`

Det finns även ett i kodskelett `given_files/uppgift3.cc` som Spock vill att du utgår från samt två test filer `given_files/CRYPTO` och `given_files/SPOCK` med testmeddelanden.

KRAV: Uppgiften ska lösas med hjälp av standardbiblioteket och dess algoritmer och behållare.

KRAV: Filnamnet ska läsas in via kommandoraden. Antalet argument ska kontrolleras och vid fel ska programmet avsluta och skriva ut ett felmeddelande till `cerr`.

KRAV: Du får använda en manuell loop **eller** en `std::for_each()` i programmet.

Uppgift 4 - Smartare pekare till kort

Riker är en vardaglig kortspelare och har nu fått ett intresse för att programmera ett kortspel. Han är en flitig programmerare men avskyr manuell minneshantering och skulle därför vilja ha en smartare pekare till sina kort som kan ta hand om minneshanteringen. Han har därför bett dig att implementera just en sådan pekare.

Kraven Riker har för denna kortpekare är att den ska kunna hantera minnet själv, dvs den allokerar minnet för kortet vid skapandet av kortet samt avallokerar minnet då kortpekaren destrueras. Kortpekaren ska inte heller gå att kopiera utan endast flytta på, Riker vill inte att någon ska kunna fuska och kopiera kort. Slutgiltiga funktionaliteten som pekaren måste ha är att den måste kunna bli avrefererad. Detta betyder att du måste implementera `*` operatorn för kortpekaren och returnera rätt värde då avreferering sker på kortpekaren. Tänk på att det ska vara samma data pekaren pekar på och inte en kopia. Deklarationen av överlagringen bör se ut så här:

```
Card& operator*();
```

Det finns ett kort testprogram i filen `given_files/uppgift4` tillsammans med en implementation av kortet. Du kan behöva lägga till testfall för att säkerställa funktionaliteten för pekaren samt att minneshanteringen är korrekt, dvs att det inte finns några minnesläckor.

KRAV: Programmet skall **inte** läcka minne

KRAV: Du får **inte** använda `std::unique_ptr` eller `std::shared_ptr` för den här uppgiften