TDP004 – Tenta

2025-08-27
14:00–19:00

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.

Hjälpmedel	En C++-bok (t.ex. Stroustrup) Ett A4-ark med egna anteckningar

Examination

Tentamen består av två delar, Del I och Del II. Båda bedöms under tentamens gång och återkoppling ges. Detta ger möjligheten att korregera en uppgift och skicka in igen.

Alla uppgifter i Del I ges betyget *Godkänd*, *Kompletteras* eller *Underkänd*. Betyget *Godkänd* sätts på en uppgift när uppgiften är komplett och uppfyller alla specifierade krav. Betyget *Kompletteras* sätts om det finns vissa problem som behöver åtgärdas. Då kan uppgiften skickas in igen. Betyget *Underkänd* ges om inga framsteg gjort sedan förra inlämningen. Då finns det ingen möjlighet att skicka in uppgiften igen.

Uppgifterna i Del II bedöms med maximalt 10 poäng vardera. De kan skickas in igen så länge *signifikanta* förbättringar görs. Om inga signifikanta förbättringar görs ges en sista chans att skicka in uppgiften, sedan låses uppgiften till den uppnådda poängen.

Betygskriterier

För alla betyg måste alla uppgifter i Del I vara godkända *och* poäng från Del II behöver samlas in enligt följande tabell:

Insamlade poäng	**Betyg**
10p	3
17p	4
24p	5

Bonuspoäng

Bonus gäller endast under dugga samt första ordinarie tentamen i samband med kursen.

För varje av kursens 6 laborationer som lämnades in i tid och blev godkänd med max en komplettering ges 0.5 bonuspoäng i Del II. Maximalt antal bonuspoäng är därmed 3. Inga halva poäng ges ut.

Del I avklarad under duggan

Om du klarat av Del I i sin helhet under duggan och det är *första ordinarie tentamen* efter duggan så får du Del I avklarad på tentamen. Detta förs in i systemet manuellt efter tentamen börjat och kommer därför ta ett tag innan det dyker upp för dig i klienten.

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.

Del I

I denna del ska du lösa tre uppgifter. Din lösning på dessa uppgifter måste uppfylla alla specificerade krav, följa god programmeringsed och vara korrekt C++-kod.

Uppgift #1 – STL

I den givna filen `assignment1.cc` finns en `struct` som representerar en anställd på ett företag. Den ser ut som nedan. Det finns också en lista av anställda sparade i en vektor.

```
struct Employee
{
    string name;
    int salary;
};
```

Skriv ett program som skriver ut medianvärdet av de anställdas lön.

Tips: Median är det värde som skulle ligga i mitten av mängden om mängden var sorterad.

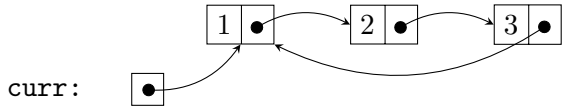
Funktionella krav: Körning av programmet ska ge följande utdata i terminalen.

```
The median salary is: 10005
```

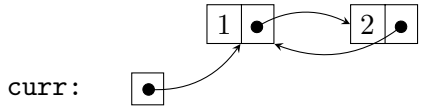
Icke-funktionella krav:

- Du måste använda en eller flera lämpliga STL-algoritmer tillsammans med lambda-uttryck för att få fram behållaren med rätt innehåll.
- Utskriften av resultatet får du göra på vilket sätt du vill. Det behöver alltså inte vara en STL-algoritm.
- Din lösning ska gå att applicera på en godtycklig vektor med denna typ av objekt.
- Du får inte ändra på den givna koden, endast lägga till mer.

Uppgift #2 – Minne



Skriv kod som skapar den länkade strukturen ovan. Skriv sedan kod som modifierar strukturen så att den ser ut som nedan. Du måste använda den givna `Node` från `assignment2.cc`.

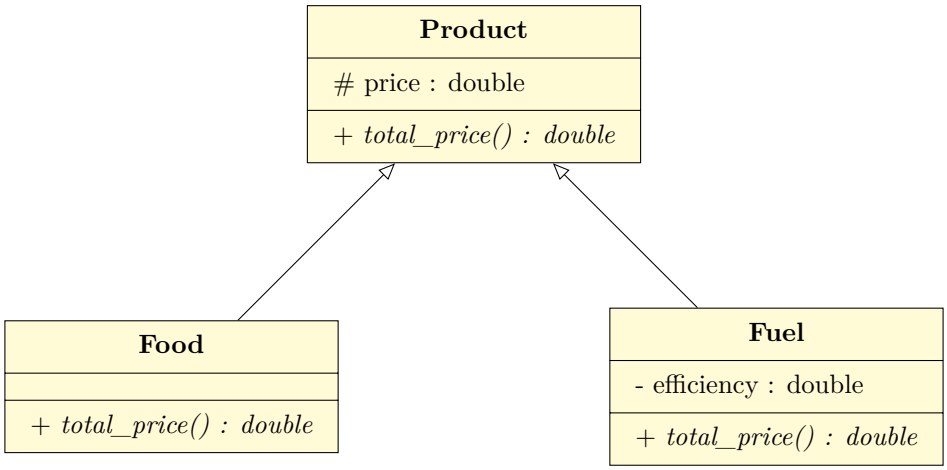


Icke-funktionella krav:

- Samtliga element ska vara dynamiskt allokerade. Du behöver endast avallokera element som kopplas bort från strukturen.
- Du får inte deklarera några fler variabler än de som redan är deklarerade i `main()`.

Uppgift #3 – Polymorfi

Implementera följande klasshierarki:



- `Product::total_price()` returnerar `price * 1.4`
- `Fuel::total_price()` returnerar `price` multiplicerat med `1.2 + 0.1 / efficiency`
- `Food::total_price()` returnerar `price * 1.1 + 20.0`

Icke-funktionella krav:

- Alla dina klasser ska vara deklarerade och definierade i *en* källkodsfil.
- Den givna filen `assignment3.cc` har ett main-program som ska fungera oförändrat.

Funktionella krav: Körning av programmet ska ge följande utdata i terminalen:

```
Total prices:
coffee: 28
pan: 278.6
electricty: 2.65
petrol: 5.6
potato: 21.65
tomato: 25.5
```

Part II

Uppgifter i Del II är värda upp till 10 poäng.

Uppgift #4 – STL

I denna uppgift ska du utveckla ett system som hjälper ett gäng med vänner som ska plannera en piknik. Varje person i gruppen ger förslag på föremål som ska tas med och programmet ska då skriva ut en lista på de viktigaste föremålen som har föreslagits. Men det finns ett antal komplicerande faktorer:

- Vissa individer skriver föremålen med blandade gemener och versaler (små- och stora bokstäver) och andra skriver det med endast gemener, vilket betyder att varje föremål måste standardiseras till ett enhetligt format.
- Flera personer föreslår samma föremål, dessa föremål ska endast förekomma en gång i den genererade listan av föremål.
- Den genererade listan ska skrivas ut i sorterad ordning efter längden av föremålens namn (föremål med korta namn ska förekomma först).
- Om två föremål har lika långa namn ska dessa internt sorteras i alfabetisk ordning.
- Endast *hälften* av alla föreslagna föremål kan tas med, därför ska endast den *första* halvan av de sorterade föremålen skrivas ut.

Programmet ska läsa in alla personers förslag som strängar från `std::cin` till en behållare. Användaren signalerar att inmatningen är klar genom att trycka `<enter>` tangent följt av att trycka `ctrl+D`. Du måste skriva ditt program så att alla krav ovan uppfylls och skriver ut den slutgiltiga listan i enlighet med körexemplen nedan.

Krav: Inga loopar, rekursion eller `std::for_each` får användas för att lösa denna uppgift.

Krav: Du får endast skapa en *behållare*, specifikt den behållare som innehåller alla förslag, så den slutgiltiga listan av föremål ska förekomma i samma vektor i slutet av programmet.

Funktionella krav: Körning av testprogrammet ska ge följande utdata i terminalen, användarinmatning är i fetstil.

```
$ ./a.out
Apple banana sandwich Chips apple Juice banana napkin
apple chips juice
```

```
$ ./a.out
lemonade sandwiches FRUITS Fruits Sandwiches fruits Lemonade SANDWICHES CHIPS
chips fruits
```

Uppgift #5 – Minne

I `assignment5.cc` finns en delvis implementerad list-klass. I denna uppgift ska du färdigställa implementationen genom att skriva klart `insert()`, implementera destruktorn samt lägga till den nya funktionen `zipmerge()`.

Funktionen `zipmerge()` tar in en `List` vid namn `other` och returnerar en helt ny lista där elementen från `*this` och `other` är sammanflätade.

Exempel: Om vi har två listor, `L1` som innehåller `{1,2,3}` och `L2` som innehåller `{4,5,6}`, så skulle `L1.zipmerge(L2)` ge ut listan `{1,4,2,5,3,6}`.

Beteendet är odefinierat om längderna av de två listorna skulle vara olika. Det innebär att vad som helst får hända, men programmet får **inte** krascha.

I denna uppgift bryr vi oss inte om de andra speciella medlemsfunktionerna. Du behöver alltså inte implementera kopierings- eller flyttsemantik.

Ickefunktionella krav:

- Programmet ska köra utan minnesläckor.
- Du får inte modifiera den givna `main()`-funktionen.

Funktionella krav: Körning av testprogrammet ska ge följande utdata i terminalen.

```
1 3 5 7
2 4 6 8
1 2 3 4 5 6 7 8
```

Uppgift #6 – Polymorfi

Olika pjäser i schack har olika regler för hur de får flytta sig. I denna uppgift ska du implementera klasser för att representera tre av dessa pjäser. En schackpjäs ska representeras av en koordinat i x- och y-led i intervalet 0 till 7. Varje klass ska implementera funktionalitet för att

- kolla om pjäsen från sin nuvarande position kan flytta till en annan angiven position.
- kolla om pjäsen kan ta en annan pjäs.

Utöver de tre klasserna nedan behöver du fylla i luckorna i det givna testprogrammet och implementera klasser för att testprogrammet ska fungera och för att unvika koduppprepning.

Pawn (Bonde): En bonde får endast flytta ett steg framåt. Den kan inte flytta åt sidan eller bakåt. För att hålla koll på vilket håll den flyttar sig ska klassen ha en variabel som håller koll på detta. Den implementeras som ett heltal med värde 1 (för vita pjäser) eller -1 (för svarta pjäser). Givet en position `new_x` och `new_y` så kan vi kontrollera om bonden kan flytta dit med

```
(old_x == new_x) && (old_y + direction == new_y)
```

En bonde kan ta pjäser som står ett steg framåt och ett steg åt sidan (alltså diagonalt framåt). I exemplet nedan kan bonden på position (1,1) ta tornet på position (2,2). Den skulle dock inte kunna ta en pjäs som står på position (1,2) eller (2,1).

Rook (Torn): Ett torn kan flytta ett obegränsat antal steg i x-led eller y-led, men inte diagonalt. Detta kan kontrolleras med

```
(old_x == new_x) != (old_y == new_y)
```

Ett torn kan ta en pjäs om den kan flytta till positionen den andra pjäsen står på.

Queen (Dam): En dam kan flytta sig som ett torn. Utöver det kan den flytta sig diagonalt i alla fyra riktningar. Detta kan kontrolleras med

```
(old_x == new_x) != (old_y == new_y) || abs(old_x-new_x) == abs(old_y-new_y)
```

En dam kan ta en pjäs om den kan flytta till positionen den andra pjäsen står på.

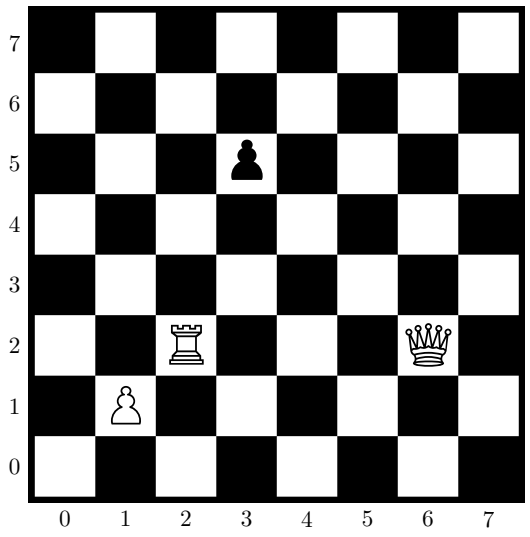


Figure 1: En illustration av schackpjäsernas placering i testprogrammet.

Krav på nästa sida!

Tips: Detta är inte en 100% korrekt implementation av schack med alla regler inkluderade. Implementera ditt program enligt de regler som står skrivna så att testprogrammet fungerar. Att implementera ett helt korrekt schack är långt utanför uppgiften.

Icke-funktionella krav:

- Du ska strukturera dina klasser så att koduppprepning undviks.
- Ditt program ska inte resultera i minnesläckor.

Funktionella krav: Körning av testprogrammet ska ge följande utdata i terminalen.

```
== White Pawn ==
true true
== Black Pawn ==
true true
== Rook ==
true true
== Queen ==
true true
== Can Attack ==
true true true true
```