TDP004 – Tenta

2025-01-17
08:00–13:00

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 `uppgift1.cc` finns en `struct` som representerar brädspel och en behållare med brädspel som representerar Pontus brädspel.

```
struct Game
{
    string name;
    int fun;
    int storage;
};
```

Skriv kod som kontrollerar att brädspelssamlingen är optimal. Om den inte är det så ska namnet på alla icke-optimala brädspel skrivas ut. Om den är optimal så görs ingen utskrift. Ett brädspel är optimalt om `fun` delat på `storage` för ett spel är större eller lika med `3`.

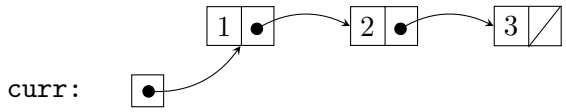
Funktionella krav: Körning av programmet ska ge följande utdata i terminalen.

```
Descent Journeys in the Dark: first edition
Arcs
```

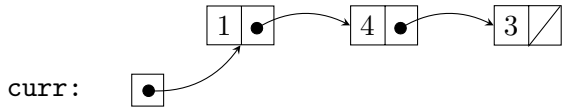
Icke-funktionella krav:

- Du måste använda en eller flera lämpliga STL-algoritmer tillsammans med lämpliga lambda-uttryck så att din lösning går att applicera på en godtycklig vektor med spel.
- 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.

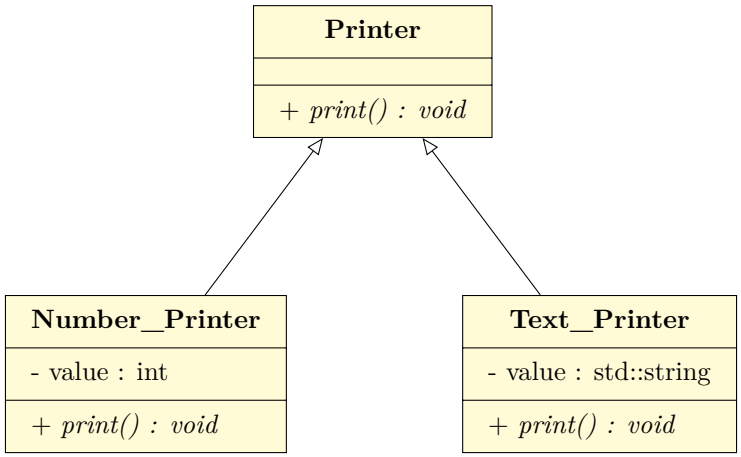


Icke-funktionella krav:

- Samtliga element ska vara dynamiskt allokerade och koden får inte generera minnesläckor.
- Max 2 variabler i ditt program får spara data av pekartyp
- I slutet av programmet ska det finnas en variable **curr** som pekar på det första elementet i strukturen.

Uppgift #3 – Polymorfi

Implementera följande klasshierarki:



`Number_Printer::print()` skriver ut heltalet `Number_Printer::value` till `std::cout`.

`Text_Printer::print()` skriver ut strängen `Text_Printer::value`, omsluten av `"` tecken, till `std::cout`.

Du måste också skapa en konstruktor för `Text_Printer` och `Number_Printer` som initierar alla datamedlemmar i vardera klass.

Icke-funktionella krav:

- Alla dina klasser ska vara deklarerade och definierade i *en* källkodsfil.
- Den givna filen `uppgift3.cc` har ett main-program som ska fungera oförändrat.

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

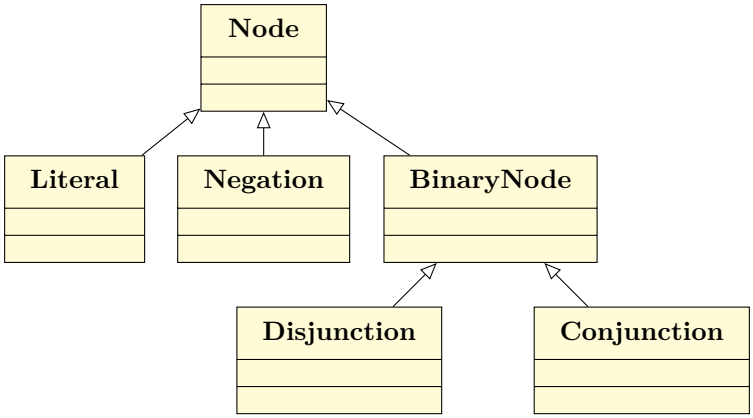
```
5
"My text"
```

Part II

Uppgifter i Del II är värda upp till 10 poäng.

Uppgift #4 – Polymorfi

Träd är ett vanligt sätt att representera logiska och aritmetiska uttryck på i programmering. I denna uppgift ska du utöka den givna koden i `uppgift4.cc` med de klasser som krävs för att programmet ska fungera som förväntat, ett påbörjat uml-diagram finns nedan:



En kort beskrivning av varje klass, du behöver också härleda viss information från huvudprogrammet och det du kan om objektorienterad design och principer:

Node Basklass i hierarkin. Alla noder som har ett specifikt beteende för medlemsfunktionen `eval()` behöver implementera funktionen `eval()`.

Literal Lagrar en `boolean`. Vid anrop till `eval()` returnerar objekt av denna klass det lagrade sanningsvärdet.

Negation Lagrar en referens eller pekare till ett barn. Vid anrop till `eval()` utvärderar objekt av denna klass sitt barn och returnerar inversen av det sanningsvärde som barnet returnera.

BinaryNode En nod som har referenser eller pekare till två andra noder.

Disjunction Vid anrop till `eval()` returnerar objekt av denna klass resultatet av att anropa `eval()` på båda barnen och applicera logisk `or` på deras returvärden.

Conjunction Som `Disjunction` men med logiskt `and`.

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

```
true
```

Uppgift #5 – STL

I denna uppgift ska du skapa ett program som håller koll på hur ett butikslager ser ut. I filen `stock.cc` finns det en `struct` vid namn `Product` som innehåller tre fält: namnet på produkten (`name`), hur mycket en enhet av produkten kostar (`price`) och hur många enheter som finns i lager (`stock`).

Din uppgift är att:

1. modifiera den givna funktionen `read_products()` så att den inte längre använder några loopar. Denna funktion anropas i huvudprogrammet där resten av stegen kommer att utföra operationer på vektorn av produkter som returneras.
2. Ordna vektorn så att den partitioneras i två delar: första delen innehåller alla produkter som finns i lager (d.v.s. alla produkter som har `stock` större än noll) och andra delen innehåller de produkter som **inte** finns i lager.

   Varje del måste internt vara sorterad på ett sådan sätt att den dyraste produkten ligger först och den billigaste längst bak.

   Notera: Båda delarna ska fortfarande lagras i `products` vektorn, du måste själv hålla koll på vart i vektorn som andra delen börjar.
3. Skriv sedan ut de två delarna i enlighet med körexemplet nedan.

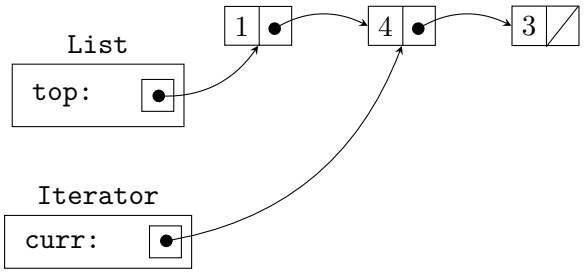
Ickefunktionella krav: I denna uppgift ska du använda algoritmer från STL. Inga loopar eller rekursion tillåts. Du får inte skapa några nya behållare, utan allt arbete ska utföras på den givna vektorn.

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

```
Products in-stock:
 - Fryer (249 SEK)
 - Mat (199.5 SEK)
 - Mop (129.9 SEK)
 - Broom (49.9 SEK)
 - Soap (39.9 SEK)
 - Wrap (29.5 SEK)
 - Bags (25.5 SEK)
 - Towel (19.8 SEK)
 - Brush (15 SEK)
 - Sponge (12.5 SEK)
Product out-of-stock:
 - Iron (399 SEK)
 - Pot (299.5 SEK)
 - Trashcan (149 SEK)
 - Toilet (99.9 SEK)
 - Laundry (89.9 SEK)
 - Vacuum (79 SEK)
 - Cloth (79 SEK)
 - Cleaner (45 SEK)
 - Detergent (35.5 SEK)
 - Spray (25 SEK)
```

Uppgift #6 – Minne

En iterator är en klass som håller koll på en specifik plats i en behållare. Hur detta implementeras beror på vilken behållare som används. I vårt fall kommer det vara en länkad lista. Då håller vi koll på en specifik plats genom en pekare till den nod som håller värdet. I bilden visas ett iteratorobjekt som pekar ut det andra värdet i en lista.



I denna uppgift ska du utöka en given implementation av en länkad lista med en enkel variant av en iterator [1]. Du behöver modifiera den givna klassen `List` och skriva en egen klass `Iterator`.

Klassen `Iterator` ska internt lagra en pekare till en `Node` i listan. Den implementerar en lämplig konstruktor samt följande medlemsoperatorer.

- Avrefereringsoperator (`int& operator*()`) - Funktionen ska ge tillbaka en referens till det värde som iteratorn interna pekare just nu håller koll på.
- Stegningsoperator (preinkrement) - Funktionen ska stega iteratorn ett steg framåt i listan och sedan returnera en referens till det egna objektet. (postfix-varianten av operatorn behöver inte implementeras)
- Jämförelseoperator för likhet - Funktionen ska avgöra hurvida två iteratorer pekar på samma plats. Detta görs genom att jämföra adresserna som de två iterator-objekten sparar.

Utöver detta behöver du implementera följande funktioner i klassen `List`.

- `begin` - Funktionen ska skapa och returnera ett Iteratorobjekt som initieras med en pekare till det första elementet i listan.
- `end` - Funktionen ska skapa och returnera ett Iteratorobjekt som initieras med en `nullptr`.

I denna uppgift tar vi inte hänsyn till de övriga speciella medlemsfunktionerna. Du behöver alltså inte implementera flytt och kopiering för någon av klasserna.

Ickefunktionella krav:

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

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

```
1 12 7 2 3
```

[1] Detta blir inte en riktig iterator utan ett urval av de funktioner som en iterator från standardbiblioteket har. Typiskt ligger en iterator som en intern klass i behållaren den hör till. I denna uppgift implementerar vi iteratorn som en klass utanför behållaren för att slippa cirkulära beroenden.