TDP004 – Tenta

2025-03-20
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 `uppg1.cc` finns en `struct` som representerar en bok och en behållare med böcker som representerar Erics boksamling.

```
struct Book
{
    string name;
    int pages;
};
```

Skriv kod så att vi i slutänden av programmet har en lämplig behållare som endast innehåller alla böcker som är kortare än 294 sidor. Skriv sedan ut den behållaren.

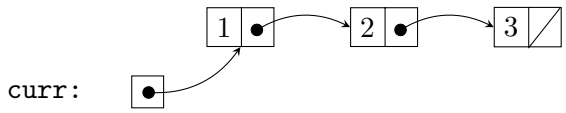
Funktionella krav: Körning av programmet ska ge följande utdata i terminalen.

```
Mysteries of the Deep: 250
Whispers in the Wind: 230
Journey to the Unknown: 293
```

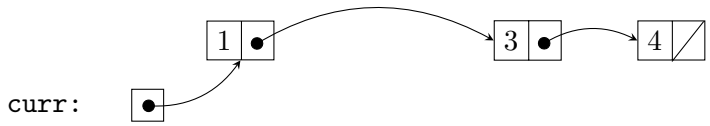
Icke-funktionella krav:

- Du måste använda en eller flera lämpliga STL-algoritmer tillsammans med lambda-uttryck för att få fram behållaren med rätt innehåll.
- Utskriften av listan 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 `uppg2.cc`.

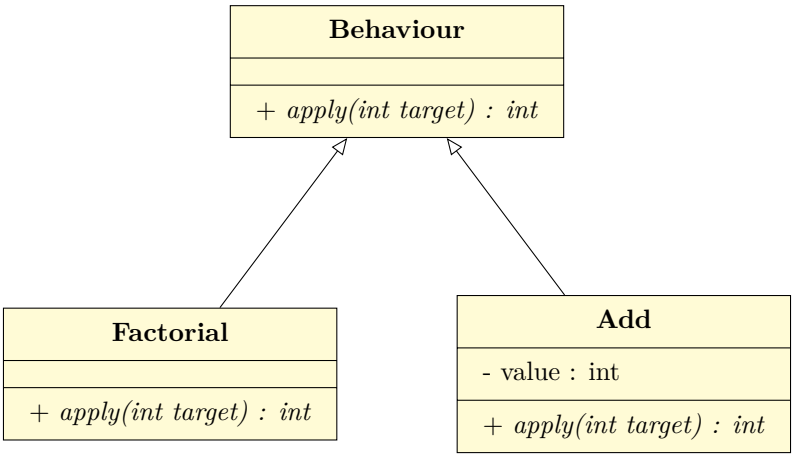


Icke-funktionella krav:

- 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:



- `Behaviour::apply(int target)` har inget definierat beteende.
- `Factorial::apply(int target)` returnerar fakulteten av `target`[1]
- `Add::apply(int target)` returnerar summan av `target` och datamedlemmen `value`

Icke-funktionella krav:

- Alla dina klasser ska vara deklarerade och definierade i *en* källkodsfil.
- Den givna filen `uppg3.cc` har ett main-program som ska fungera oförändrat.

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

```
5040
```

[1] Att beräkna fakulteten räknar vi som ett förkunskapskrav. Men om du inte klarar av att göra det så kan du istället approximera fakulteten med $target * target$. Du kan då förvänta dig resultatet 100.

Part II

Uppgifter i Del II är värda upp till 10 poäng.

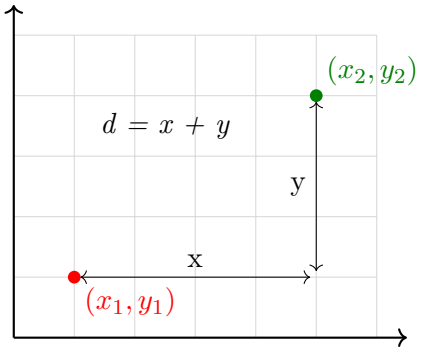
Uppgift #4 – STL

I denna uppgift ska du skapa ett program som beräknar medellängden av avstånden från en given startpunkt till en lista av punkter. I filen `points.txt` finns listan av punkter med en punkt på varje rad.

Först behöver du skapa en lämplig datatyp för att representera en punkt med tillhörande operationer som är relevanta för att lösa problemet nedan.

Programmet ska utföra följande steg, i ordningen de står.

1. Läsa in punkterna från filen till en lämplig behållare, `point_list`. Behållaren ska internt spara objekt av din användardefinerade datatyp.
2. Läsa in startpunkten från terminalen. Punkten anges som två heltal separerade med ett mellanslag. Se körexempel nedan.
3. Skapa en ny behållare, `distance_list`, som sparar flyttal.
4. Fyll `distance_list` med avstånden från startpunkten till respektive punkt i `point_list`. Avståndet beräknas som "manhattanavståndet", dvs att man summerar distansen i x-led och distansen i y-led.



5. Räkna ut summan genom att addera ihop alla värden i `distance_list` och skriv sedan ut medelvärdet genom att dividera summan med antalet värden i `distance_list`

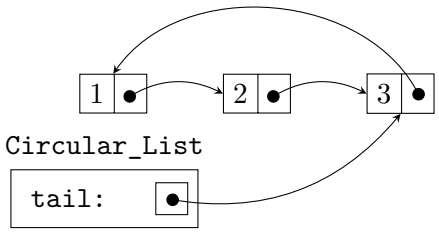
Ickefunktionella krav: I denna uppgift ska du använda algoritmer från STL. Inga loopar, rekursion eller `std::for_each` tillåts.

Funktionella krav: Körning av programmet ska ge följande utdata i terminalen. (Text i fetstil markerar användarindata.)

```
Mata in en punkt: 3 3
Medellängden från din punkt är: 9.57143
```

Uppgift #5 – Minne

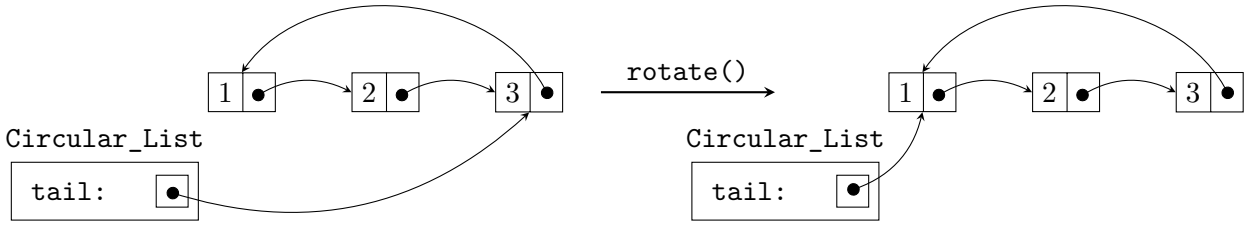
Cirkulärt länkade listor är en speciell variant av länkade listor där sista noden alltid länkar tillbaka till första, vilket skapar en ”oändlig” loop av noder. Nedan ser du en bild på ett exempel av en cirkulärt länkad lista. Notera att eftersom att listan är cirkulär kan vi välja att lagra pekaren till första eller sista noden i sekvensen. I denna uppgift har vi valt att lagra pekaren till det *sista* elementet.



I filen `uppg5.cc` finns en påbörjad implementation av en cirkulärt länkad lista.

Din uppgift är att:

- Lägg till allokering av noden i `push_front()`
- Lägg till avallokering av noden i `pop_front()`
- En destruktor som avallokerar hela listan.
- Funktionen `front()` som returnerar det värde som för tillfället ligger i början av listan.
- Funktionen `rotate()` som *roterar* hela listan ett steg. Detta innebär att elementet som låg först i listan innan anropet till funktionen hamnar nu sist, och det element som låg näst längst fram ligger nu först o.s.v. Se följande diagram:



  Om listan är tom görs ingenting.
- Funktionen `print()` som tar emot en ström och skriver ut värdena i listan på den strömmen. Denna funktion får inte göra några som helst ändringar av strukturen.

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

Ickefunktionella krav:

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

Notera: I testfallen används `assert()` vilket är ett enkelt sätt att göra tester på (tänk en simplare variant av `catch.hpp`). Om programmet gör en utskrift indikerar det att någonting inte fungerar enligt specifikation vilket kommer leda till poängavdrag.

Uppgift #6 – Polymorfi

I ett nytt äventyrsspel med 2D grafik ska en spelare kunna röra sig runt och interagera med objekt i spelet. I spelet ska det finnas minst två typer av spelobjekt som påverkar spelaren. Spelaren representeras av `struct Player` som finns given i filen `uppg6.cc`. där finns även ett huvudprogram för att testa de skapade spelobjekten. Varken spelaren eller huvudprogrammet får modifieras. Alla positioner i spelet är heltal.

Utöver basklassen som representerar spelobjekt (`GameObject`), som du designar själv, ska du implementera två klasser:

Portal: En portal har en position på spelplanen och äger (ansvarar för) ett annat spelobjekt som fungerar som portalens destination. På ett portalobjekt ska det gå att anropa tre medlemsfunktioner:

- `void move_player_here(Player& p)`: Sätter spelarens position till portalens position.
- `bool collides(Player const& p) const`: Returnerar sant om spelarens position är samma som portalens position.
- `void affect(Player& p) const`: Sätter spelarens position till destinationens position ***och*** låter sedan destinationen utföra sin påverkan på spelaren.
- Kopieringskonstruktor och kopieringstilldelsningsoperator tas explicit bort (eller implementeras korrekt) eftersom denna klass ansvarar för ett annat objekt. Borttagning av en speciell medlemsfunktion görs genom `= delete`.

Bouncer: En studsare har en position på spelplanen och håller reda på sin studsförmåga (ett heltal). På ett studsobjekt ska det gå att anropa tre medlemsfunktioner:

- `void move_player_here(Player& p)`: Sätter spelarens position till studsarens position.
- `bool collides(Player const& p) const`: Returnerar sant om spelarens position är samma som studsarens position.
- `void affect(Player& p) const`: Subtraherar spelarens hastighet $v = (vx, vy)$ multiplicerat med studsförmågan f från spelarens position $p = (x, y)$. Dvs $(p = p - v * f)$.

Icke-funktionella krav:

- Det ingår i uppgiften att undvika koduppprepning. Gör lämplig basklass `GameObject`.
- Det ingår i uppgiften att testprogrammet ska fungera. Gör lämpliga konstruktorer.
- Det ingår i uppgiften att se till att inga minnesläckor uppstår så som testprogrammet är skrivet. Gör lämplig(a) destruktor(er).

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

```
$ ./a.out
ok1
ok2
ok3
ok4
ok5
ok6
ok7
```