TDP004 – Dugga

2024-12-04
13:00–17:00

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.

Uppgiften i Del II bedöms med maximalt 10 poäng. Den 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 ett godkänt betyg måste alla uppgifter i Del I vara godkända *och* 10 poäng insamlas i Del II.

Insamlade poäng	**Betyg**
10p	3

Bonuspoäng

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

För varje av kursens 6 labborationer 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.

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 specifierade krav, följa god programmeringsed och vara korrekt C++-kod.

Uppgift #1 – STL

```
struct Cylinder
{
  double radius;
  double height;
};

int main()
{
  std::vector<Cylinder> cylinders {
    Cylinder{ 2.5, 2.3 }, Cylinder{ 3.4, 1.6 }, Cylinder{ 1.9, 4.5 },
    Cylinder{ 4.0, 1.5 }, Cylinder{ 3.2, 2.0 }, Cylinder{ 4.8, 1.1 }
  };
  double const minimum_volume {33.0};
}
```

Skriv kod som kontrollerar att varje element i `cylinders` har en volym som är större än det givna minimivärdet. Om så är fallet ska "Ja" skrivas ut, annars "Nej".

En cylinders volym beräknas med $volume = \pi * radius^2 * height$.

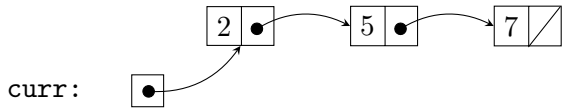
Funktionella krav: Körning av programmet ska ge följande utdata i terminalen.

```
Ja
```

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 värden.
- Du får inte ändra på den givna koden, endast lägga till mer.

Uppgift #2 – Minne



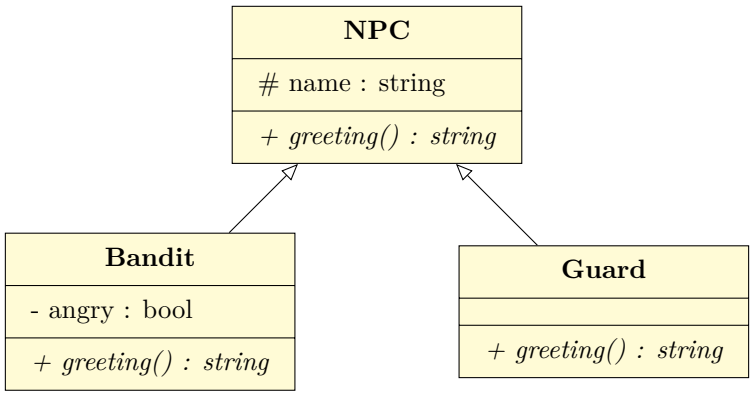
Skriv kod som skapar följande struktur. Modifiera sedan strukturen för att lägga till ett element med värde 9 mellan element med värde 5 och 7. Ditt huvudprogram måste innehålla en variabel `curr` som pekar på första elementet i den länkade strukturen. När du är klar ska `curr` fortfarande peka på samma element. Dina element ska deklareras som ett aggregat på formatet:

```
struct Node
{
  int const val;
  Node* next;
}
```

Icke-funktionella krav: Samtliga element ska vara dynamiskt allokerade och koden får inte generera minnesläckor.

Uppgift #3 – Polymorfi

Implementera följande klasshierarki.



`greeting()` har följande beteende för respektive klass.

NPC returnerar `"Hello, my name is "` följt av dess namn.

Guard returnerar `"Stop in the name of the jarl! I am "` följt av dess namn.

Bandit göra samma sak som för `NPC` om `angry` är `false`. Annars returneras `"Never should have come here!"`.

Icke-funktionella krav:

- Alla dina klasser ska vara deklarerande 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.

```
Hello, my name is Saadia
Never should have come here!
Hello, my name is Hrongar
Stop in the name of the Jarl. I am Lydia
```

Part II

Uppgiften i Del II är värd 10 poäng.

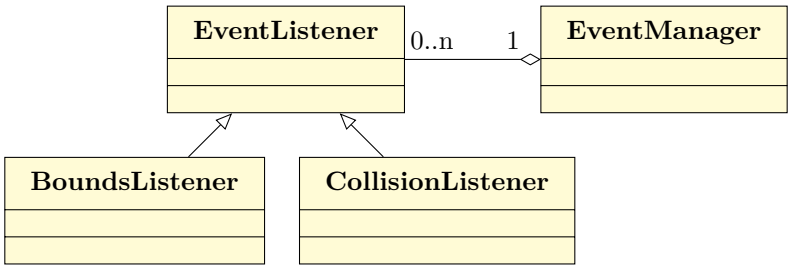
Uppgift #4 – Polymorfi

Ett vanligt problem inom programmering är att objekt A behöver reagera på när något händer i objekt B. Ett vanligt sätt att lösa detta är genom Observer-möstret.[1] I detta mönster skapas en `EventManager` som har en lista av alla objekt som är intresserade av en specifik händelse. När en händelse kommer in till `EventManager` går den igenom sin lista av alla intresserade objekt och anropar en funktion på varje objekt lagrat i listan.

Funktionen som anropas specifieras i en gemensam basklass för alla typer av objekt som kan reagera på en händelse – här kallat `EventListener` – som specifierar ett gränssnitt för alla som ska lyssna efter händelser.

Utöver de två generella klasserna ska du implementera två specifika lyssnare. Båda dessa lyssnar efter ändringar i en spelares position. Spelaren rör sig endast i x-led. Positionen skickas in till lyssnarfunktionen som ett heltal.

- `BoundsListener` ska kontrollera att den nya positionen är inom sitt specifierade interval och skriva ut på given ström om spelaren går utanför intervallet.
- `CollisionListener` håller koll på en position och ska kontrollera om den nya spelarpositionen gör att nya spelarpositionen är identisk med lyssnaren position. Den ska även räkna antalet kollisioner som skett.



I `uppgift4.cc` finns ett givet huvudprogram. Din uppgift är att utöka denna fil med de fyra klasser som beskrivs i texten. Relationen mellan dessa klasser anges i diagrammet ovan. Du behöver själv lista ut vilka funktioner och datamedlemmar respektive klass behöver för att lösa uppgiften.

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

```
Collision at 3
 - Total collisions with 'rock' is now 1
Player out of bounds at 8
Collision at -1
 - Total collisions with 'tree' is now 1
Collision at 3
 - Total collisions with 'rock' is now 2
Player out of bounds at -7
Collision at 3
 - Total collisions with 'rock' is now 3
```

[1] I uppgiften implementeras en förenklad variant av Observer-mönstret.