Duggainformation

Duggan har två syften:

- Duggan tjänar som inträdesbiljett att få påbörja projektet. Duggan är den tekniska intervjun i rekryteringen av projektgruppsmedlemmar.
- Duggan tjänar som individuell examination av kursmål för betyg 3.

Duggan har en uppgift och bedöms G eller U. Vi kräver tydlig, lättfattlig och underhållsbar kod, men vi kommer *inte* vara lika petiga som i labserien. En godkänd lösning ska fungera korrekt, lösningen använder lämplig klasshierarki, den saknar minnesläckor, den undviker koduppprepning, och lösningen har någorlunda namngivning och indentering.

Duggatid

Du har 120 minuter på dig att lösa duggan. Du med rätt till förlängd skrivtid har 160 minuters skrivtid. Rätt till förlängd tid beslutas av LiU centralt.

Ämnesinnehåll

Fokus på duggan kommer vara att skapa en minimal klasshierarki och få den att lösa en uppgift. Detta medför att god kunskap om syntax och semantik rörande klasser behövs.

1. Grunder: typer, variabler, satser, std::vector, funktioner, parametrar, argument
2. Klass: objekt/instans, konstruktor, inkapsling, datamedlem, medlemsfunktion, public, private
3. Arv: basklass, härledd klass, anrop av basklasskonstruktorkonstruktor, protected, anrop av basklassfunktion
4. Polymorfi: slicing, virtuell destruktor, dynamisk/statisk bindning, virtual, pure virtual, override
5. Minneshantering: allokering, avallokering, minnesläcka

Hjälpmedel

Under duggan råder standard tentamensregler. Du får ha med följande hjälpmedel:

- En bok om C++. För boken gäller följande regler:
  - Kommentarer eller noteringar som direkt rör text och exempel på sidan i fråga får finnas i sidmarginalen.
  - Egna sidflikar för att enkelt kunna hitta t.ex. de olika kapitlen är tillåtna.
  - Inga extra ark eller lappar, lösa eller fastsatta, får finnas.
  - Ingen programkod på tomma sidor, insidan av pärmarna, försättsblad, etc.
- Ett A4-ark. Valfritt innehåll på vardera sida. Går ej ersätta med två enkelsidiga ark.
- Penna för att anteckna. Du kan be tentamensvakt om kladdpapper.

Följande får INTE tas med:

- Elektroniska prylar. Till exempel miniräknare, mobiltelefon, hörlurar, tangentbord, smartklocka etc.

Inlogging

Tentamensvakt kontrollerar anmälan vid inpassage till salen. När datorns inloggningsskärm visar "tentamensläge aktivt" kan du logga in som vanligt med ditt LiU-ID och ditt privata lösenord och börja förbereda din programmeringsmiljö. Tentamensvakten kommer under duggan gå runt för att kontrollera legitimation och hjälpmedel.

Alias för kompilering

Under duggan finns det tre alias att använda sig av för kompilering med c++17:
`w++17` Rekommenderas!
`e++17` Kompilering med alla varningar som fel.
`g++17` Kompilering utan varningar.

Frågor

Frågor ska ställas via studentklienten. Detta för att vi ska ha en historik av konversationen samt för att vi ska kunna ge samma hjälp till olika studenter. Räck upp handen om klienten, tangentbordet, musen eller datorn inte verkar fungera som de ska.

C++ referens

Det finns tillgång till valda delar av `cppreference.com`. Du måste starta webbläsaren via skrivbordsikonen "Web access" för att komma åt sidan.

Inlämning

När du har en lösning som kompilerar utan varningar, kör utan minnesläckor och löser uppgiften skickar du in den via studentklienten. Din inlämning bedöms direkt[1] och du får svar via studentklienten. Om du får betyg G är det bara att spara, avsluta, logga ut, och framåt kvällen fira! Om du får komplettering läser du noga vad som inte fungerar, fixar till det efter bästa förmåga, och skickar in igen.

Utloggning

När du är nöjd med ditt betyg (som står i studentklienten) kan du avsluta duggan. Stäng alla program och spara alla filer. Sedan loggar du ut som vanligt i menyn. Lämna inte din plats innan vanliga inloggningsskärmen syns.

[1] Beroende på mängd samtidiga inskickningar kan det ta flera minuter att få svar, räck upp handen om du inte fått svar på 15 minuter.

Uppgift

Tillståndsmaskiner är en abstrakt modell som kan användas inom datorvetenskapen. En sådan maskin består av ett antal tillstånd. Ett tillstånd kan påverkas av in-signaler. Varje giltig in-signal leder till antingen ett tidigare tillstånd, samma tillstånd eller ett nytt tillstånd.

Nu ska vi bygga en tillståndsmaskin för ett litet spel. Målet är att "motorn" eller "huvudloopen" som driver spelet ska se likadan ut oavsett vilken del av spelet vi är i. I denna uppgift kommer spelet ha två tillstånd: antingen kör vi meny-systemet eller så kör vi själva spelet. Huvudprogrammet har en stack med det aktiva tillståndet överst, och huvudloopen gör två saker i sekvens med det aktiva tillståndet: (1) ber tillståndet skriva ut ledtext till användaren och (2) låter tillståndet reagera på användarindata. I steg (2) returnerar tillståndet adressen till ett nytt tillstånd, med tre betydelser:

- **nullptr**: Ber huvudloopen avsluta nuvarande tillstånd och återgå till föregående tillstånd på tillståndsstacken.
- **this**: Ber huvudloopen behålla detta tillstånd som det aktiva.
- **new ...**: Ber huvudloopen lägga det nya tillståndet på stacken och göra det till det aktiva.

På detta vis kan varje tillstånd styra vad som ska vara nästa tillstånd, och användningen av en stack gör att vi får ett naturligt beteende på programmet. Från start lägger vi menyn som aktivt tillstånd, när användaren väljer att starta spelet skapar menyn speltillståndet, och när speltillståndet är slut återgår vi till menyn. Vi har goda förutsättningar att lägga till tillstånd så att spelet t.ex. kan byta till ett paustillstånd som sedan återgår till speltillståndet, eller att menyn kan byta till ett undermenytillstånd eller highscoresidetillstånd som återgår till menyn.
Din uppgift är att implementera två tillstånd och deras basklass:

Menu: Menytillståndet skapas med en speltitel som egenskap och har två funktioner:

- `print()` Skriver ut menyn. Det finns två punkter i menyn: *"1 Play"* och *"2 Quit"*.
- `handle()` Tar emot en sträng med användarindata (menyvalet). En 1:a startar spelet genom att returnera ett nytt speltillstånd. En 2:a avslutar menyn genom att returnera `nullptr`. Allt annat skriver ut *"Bad input, Please enter 1 or 2."* på `std::cerr` och vi stannar i menytillståndet genom att returnera `this`.

GuessTheWord: Detta är speltillståndet för ett spel som går ut på att gissa ett ord. Tillståndet skapas med det ord som är rätt svar och hur många försök användaren får.

- `print()` Skriver ut *"Guess the word, attemt N:"* där N är hur många försök som är kvar.
- `handle()` Tar emot en sträng med användarindata (gissningen).
  Om gissningen är rätt skrivs *"Great input, you win!"* ut och funktionen returnerar ett värde som gör att vi återvänder till menytillståndet.
  Om gissningen är fel och det var sista försöket skrivs *"End of attempts, you loose!"* ut och funktionen returnerar ett värde som gör att vi återvänder till meny-tillståndet.
  Annars skrivs *"Keep trying!"* ut och funktionen returnerar ett värde som gör att vi stannar i speltillståndet.

OBS: Basklassen är upp till dig att implementera på lämpligt sätt. Du behöver även justera huvudprogrammet i `given_files/gamestate.cc` (kopiera det först till skrivbordet) så att stacken lagrar lämplig datatyp och så att alla minnesläckor åtgärdas (använd `valgrind`).

OBS: Körexempel på nästa sida!

Körexempel med användarinmatning med grön bakgrund:

```
Welcome to The Game, please select:
1 Play
2 Quit

0
Bad input, Please enter 1 or 2.
Welcome to The Game, please select:
1 Play
2 Quit

1
Guess the word, attempt 3:
Hej
Keep trying!
Guess the word, attempt 2:
Hopp
Keep trying!
Guess the word, attempt 1:
Eric
End of attempts, you loose!
Welcome to The Game, please select:
1 Play
2 Quit

1
Guess the word, attempt 3:
Klas
Great input, you win!
Welcome to The Game, please select:
1 Play
2 Quit

2
```