Linköpings universitet
Institutionen för datavetenskap (IDA)
UPP-gruppen 2016-08-26

Klockslag

Mål

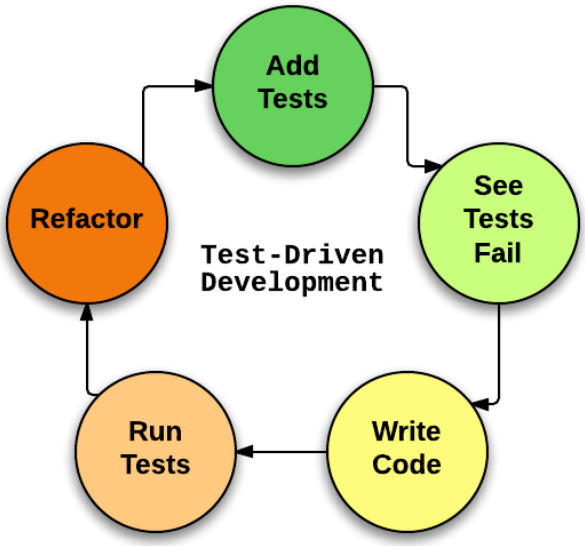
I den här laboration ska du utöver generell problemlösning lära dig att hantera sammansatta datatyper (`class`), parameteröverföring, operatoröverlagring samt testdriven utveckling (TDD, Test Driven Development).

Läsanvisningar

- Parameteröverföring (Primer kap 6.2.1-3)
- Klasser (`class`) (Primer kap 7)
- Konstruktorer (Primer kap 7.1.4)
- Åtkomst av medlemmar (Primer kap 7.2.0)
- Operatoröverlagring (Primer kap 14.1-3, 14.7)
- Dokumentationen för Catch: `https://github.com/philsquared/Catch`

Arbetssätt

När du arbetar med denna laboration ska du göra det enligt metoden TDD - testdriven utveckling. I TDD bestämmer du dig för ett litet tillägg till ditt program. Istället för att direkt börja med implementation inleds arbetet med att fundera på hur tillägget ska användas. Därefter skriver du ett program som testar funktionaliteten hos tillägget som om det redan fanns implementerat (du har alltså inte skapat det ännu). Efter att du sett att testet *inte* fungerar implementerar du ditt tillägg steg för steg tills du ser att alla test fungerar. Denna process upprepas för varje liten del av programmet så att testprogrammet blir allt större. För varje nytt tillägg körs alltså alla test, inte bara de du tänkt ut för detta tillägg. Detta för att vara säker på att din ändring inte förstört fungerande kod. Det kan också vara så att du hittar fel i dina tidigare test.



Figur 1: TDD Arbetssätt

När man skriver sitt testprogram skulle man kunna göra det med vanlig C++-kod som exempelvis if-satser och enkla utskrifter. Det kan dock bli jobbigt i längden och därför ska ni i denna laboration använda er av ett litet ramverk som heter Catch. Catch är enkelt att komma igång med och har en bra dokumentation (som anges under läsanvisningar ovan).

Uppgift: Klockslag

Du ska skapa en datastruktur och de funktioner som krävs för att kunna hantera klockslag. Ett klockslag anser vi vara tre heltal för att representera en tid på formatet `HH:MM:SS`. Åtminstone följande funktionalitet ska ni implementera för klockslag:

- Kontroll om värdet som är lagrat är ett giltigt klockslag, dvs timme är i intervallet $[0, 23]$ samt minut och sekund i intervallet $[0, 59]$.
- Möjlighet att formatera ett klockslag som en sträng. Det ska finnas möjlighet att formatera klockslaget både med 24-timmarsklocka och med am/pm-angivelse (timme 0-11 ger am, 12-23 ger pm). Resultatet ska vara en sträng på formatet `HH:MM:SS[ p]`, där `p` är pm eller am och hakparenteserna betyder att det kan utelämnas. Både värdet `"14:21:23"` och `"02:21:23 am"` ska därmed kunna genereras. En funktion `to_string()` ska returnera en sträng enligt ovan. Notera att den här funktionalitet inte ska ändra på klockslaget, utan endast hur det skrivs ut.
- Addition med ett positivt heltal `N` för att generera en ny tid `N` sekunder i framtiden med `operator+`.

```
Time t1{};
Time t2{t1 + 5};
// t2 = [00:00:05] och t1 = [00:00:00]
```

- Subtraktion med ett positivt heltal för att ge en tidigare tid med `operator-`.
- Stegningsoperatorer `operator++`, `operator--` för att stega ett klockslag en sekund. Dessa ska finnas i både prefix- och postfixversioner.

```
Time t{12, 40, 50};
t++; // t = [12:40:51]
--t; // t = [12:40:50]
```

- Jämförelse av två klockslag med de normala jämförelseoperatorerna (sex stycken), exemepelvis `operator<` och `operator==`. Fundera på om några av dessa går att implementera med hjälp av varandra.

```
Time t1{0, 0, 1};
Time t2{12, 30, 40};
t1 > t2; // detta ger 'false'
t1 != t2; // detta ger 'true'
```

- Formaterad utskrift med `operator<<`. En tidsstämpel skrivs alltid ut på formatet `HH:MM:SS`.
- Formaterad inläsning med `operator>>`. Ni kan anta att en tid alltid matas in på formatet `HH:MM:SS`. Det inmatade värdet ska dock rimlighetskontrolleras så att inte exempelvis minutvärdet är 73. Om ni upptäcker en felaktig inmatning ska strömmens `fail`-flagga sättas. Normalt sett brukar inläsningsoperatorer (endast) läsa så långt som krävs innan ett fel markeras och inläsningen avslutas, men ni väljer själva hur långt ni läser så länge det är väl dokumenterat (skriv en kommentar).

Filuppdelning och Catch

Då `catch.hpp` är ganska stor tar det ganska lång tid att hela tiden kompilera om denna. Därför rekommenderas det att du kompilerar din kod i två steg; först ett huvudprogram som säger till catch att starta och sedan länkar du ihop det med dina testfall. Här är de kommandon som krävs:

1. Kompilera filen `test_main.cc` men länka inte (skapa inte ett körbart program):
   ```
   g++17 -c test_main.cc
   ```
   Du kommer nu få en fil `test_main.o` om allt gått bra. Detta är en s.k. objektfil.
2. Du kan nu lägga till testfall i en separat fil, `time_test.cc`, och din implementation i filerna `Time.h` och `Time.cc`. När du vill kompilera ditt testprogram gör du enligt nedan:
   ```
   g++17 test_main.o Time.cc time_test.cc
   ```