DD1387 Programsystemkonstruktion med C++

Laboration 2: Kalender

2 oktober 2014

I den här labben ska du visa att du att lärt dig att använda överlagring, abstrakta basklasser och polymorfi. Ni får jobba i grupper om högst två personer. Vid redovisningen ska alla gruppmedlemmar kunna svara på frågor om alla delar av koden. Läs igenom hela lydelsen innan du börjar. För att uppgifterna ska kännas mindre lösryckta hänger de ofta ihop.

Allmänna krav:

- Ditt program ska visa att du behärskar nyckelord som `const` och `virtual` på ett korrekt sätt.
- Din kod ska vara modulariserad i klasser och filer.
- Ditt program ska inte läcka minne, så var noga med dina konstruktorer och destruktorer.
- Ditt program får använda STL.
- Innan du redovisar ska du läsa på frågeställningarna.
- Ni ska kunna redogöra för fyra olika klassdiagram och visa hur ni har använt er av polymorfi.
- Ni ska skicka in följande .h och .cpp filer till Kattis. *date, julian, gregorian, calendar, kattistime*. Dessutom testprogrammet (<labnamn>.cpp)
- Vid redovisning ska ni ha en webläsare öppen med den källkod ni skickat in till Kattis.

När labblydelsen hänvisar till filer i *kurskatalogen* avses kurshemsidan alternativt

```
/info/DD2387/kurskatalog/lab2
```

Lycka till!

2.1 (abstrakta basklasser, strikt virtuella funktioner)

Denna uppgift är den första av flera där du ska skapa datum- och kalenderklasser. Du ska hädanefter samla allt som har med din kalender att göra i namnrymden (*namespace*) lab2.

Med en abstrakt basklass definierar man upp ett gränssnitt som nedärvda klasser måste uppfylla. Skriv en generell basklass `Date` som har följande funktioner. Om inget annat sägs är returtypen `int`.

- Avgör vilka konstruktorer som ska vara med. Behövs tilldelningsoperator?

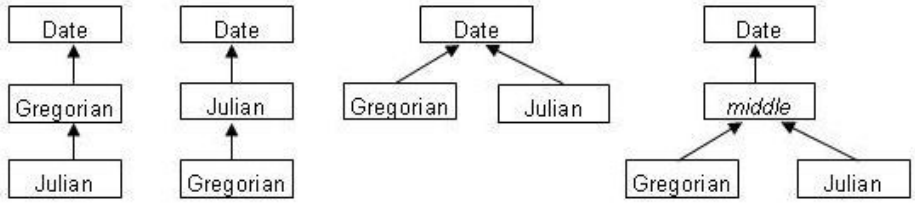
- Åtkomst: `year()`, `month()` (första månaden ska representeras av talet 1), `day()` (den första i varje månad ska representeras av talet 1), `week_day()` (måndag = 1), `days_per_week()` (returnerar ett fix antal dagar), `days_this_month()` (returnerar antal dagar den aktuella månaden) och `months_per_year()`. Skapa funktioner `week_day_name()` och `month_name()` med returtyp `std::string`.
- Mutatorer:
  - prefix-operatorer `++` och `--` som returnerar referens till sig själv.
  - `+=`, `-=` tar `int` som parameter. Välj lämplig returtyp.
  - `add_year(int n = 1)` och `add_month(int n = 1)` (lägger till `n` år/månad, där `n` är positiv eller negativ). Välj lämplig returtyp och motivera vid redovisning.
- **Obs!** Basklassen ska kunna hantera alla sorters kalendersystem (persiska, kinesiska m.m.). **Basklassen ska därför inte innehålla något som är specifikt för den kalender vi använder**, såsom 12 månader per år, veckodagarnas/månadernas namn eller dyl.
- Jämförelser (returnerar `bool`): `==`, `!=`, `<`, `<=`, `>`, `>=`, `-` (returnerar heltalsdifferensen mellan två datum)
- För att kunna jämföra olika datum som t.ex. ett persiskt datum med ett svenskt datum, så ska du definiera en metod mod_julian_day som returnerar antal dagar sedan det modifierade julianska dygnet vilket i vår nuvarande kalender är den 17 november 1858. Mer information om just detta dygn kan fås via google, eller wikipedia, sök på "modified julian day number".
- Man ska kunna skriva ut implementationer av datumklassen med utskriftsoperatorn. Deklarera `ostream & operator<<(ostream & os, const Date &);` . Låt utskriften vara år-månad-dag (med bindestreck).
- Operatorerna ska vara virtuella där så behövs. Låt de funktioner som inte kan definieras i basklassen vara strikt virtuella (*pure virtual*).
- Vilka funktioner är `const`? Vilka argument/returtyper är `const`? Vid redovisningen ska du övertyga handledaren att du kan använda `const` och `virtual`.

`Kuriosa`: Excel använder 1:a januari 1900 som referensdag. Om man öppnar Excel på PC, formaterar en cell till datum och matar in 60 så får man 29/2 1900. Den skottdagen har dock aldrig funnits. Jag hyser stor tillsikt att ni klarar er bättre än excelprogrammerarna.

2.2 (arv från abstrakt basklass)

Gör följande: skriv två klasser `Julian` och `Gregorian` för den julianska respektive gregorianska kalendern. Dessa ska ärva (direkt eller indirekt) från klassen

`Date` i uppgiften ovan. Ta ställning till nedanstående fyra alternativ att ärva. Vid redovisningen ska du reflektera över för- och nackdelar för alla de fyra alternativen.



Funktionalitet hos klasserna:

- Implementation av basklassens virtuella funktioner vid behov
- Konstruktor som tar år, månad, dag som int.
- Man ska kunna kopiera och tilldela dateobjekt. Skillnaden mellan datumen ska vara noll efter tilldelning/kopiering.
- postfix-operatorer `++` och `--` som returnerar kopia av sig själv.
- Konstruktion utan argument (defaultkonstruktorn) ger dagens datum. För att få dagens datum ska du inte använda systemanrop direkt utan gå via en speciell funktion, se filen *kattistime.h* på kurskatalogen.
- Korrekt hantering av skottår. I den julianska är det vart fjärde år. I den Gregorianska kalendern infaller skottdagen vart fjärde år men utgår de sekelårtal som inte är jämnt delbara med 400.
- Utskriftsoperatorn ska definieras och skriva ut datumet på formen YYYY-MM-DD där ental fylls ut med nollor (2000-01-01).
- `add_month` ska ge samma datum i nästa månad men om det inte går så ska det plussas på 30 dagar istället. Exempel 1/9 -> 1/10 och 31/5 -> 30/6. Den 31/1 kan bli 1:a eller 2:a mars beroende på om det är skottår.
- För skottdagen ska `add_year` ge sista februari: 29/2 2004 -> 28/2 2005. Anropar man `add_year` den 29/2 först med 1 och därefter med -1 så ska man hamna på 28/2. Lägger man till fyra år till skottdagen ska man hamna på skottdagen igen om det nu är skottår då: 29/2 2004 -> 29/2 2008. Hade man istället lagt till ett år i taget hade man hamnat på den 28:e
- Anropar man `add_month` med argumentet 5 så ska det vara ekivivalent med om man gör `add_month` 5 gånger. Motsvarande gäller inte alltid för `add_year` se ovan.
- Försöker man komma åt eller skapa ogiltiga dagar så ska `out_of_range` slängas.
- Använd engelska namn med små bokstäver på dagar och månader. Var noga med stavningen.
- Deklaration och implementation av klasserna ska skrivas i i separata header- respektive implementationsfiler (.h .cpp).

Alla implementationsdetaljer exempelvis om du har en hjälpfunktion `leap_year()`) ska döljas med `protected` eller `private`.

För att underlätta räkningar kan du låta datumen representeras av ett avstånd till ett fixt datum. Du behöver inte implementera en sluten formel för datumberäkning utan det är tillåtet att använda en hårdkodad (förberäknad) tabell som t.ex. innehåller avstånd till årets början. Det räcker om din kalender klarar datum från och med år 1858 fram till och med år 2558.

Prova dina klasser med filen `datetest.cpp`. Utöka testprogrammet med egna tester. T.ex.

```
Gregorian g;      // dagens datum
Julian j;         // också dagens datum
std::cout << "Today it is " << g << " gregorian and " << j << " julian";
if (g - j == 0) std::cout <<  ". It is the same date" << std::endl;
g = j;
if (g - j == 0) std::cout <<  "It is still the same date" << std::endl;
```

Vilket ger utskriften

```
Today it is 2006-08-01 gregorian and 2006-07-19 julian. It is the same date
It is still the same date
```

Dagens datum är alltså inte samma i kalendrarna. Följande gäller:

```
1/1   1900 Greg    = 20/12 1899 Jul
1/1   1900 Jul     = 13/1  1900 Greg
16/11 1858 Greg    = 4/11  1858 Jul   modified julian day -1
17/11 1858 Greg    = 5/11  1858 Jul   modified julian day  0
18/11 1858 Greg    = 6/11  1858 Jul   modified julian day  1
```

Det finns flera datumkonverterare mellan gregorianska och julianska kalendern på nätet om man vill kontrollera fler datumskillnader.

Historisk bakgrund Den Julianska kalendern infördes av Julius Caesar år 46 f.Kr. Detta år infogade Julius Caesar hela 90 dagar till den romerska kalendern för att återföra månaderna till deras rätta ställe på året med hänsyn till årstiderna. I den Julianska kalendern infinner sig skottår vart fjärde år.

På grund av tillkortakommanden i skottårsberäkningen i den Julianska kalendern hade påsken vid tiden för påven Gregorius XIII förskjutits tio dagar (förskjutningen var cirka 3/4 dag per århundrade). Gregorius ansåg att det var dags att räta upp detta och proklamerade att torsdagen den 4:e oktober 1582 skulle följas av fredagen den 15:e oktober. Det var bara katolska länder som följde påbudet. I Sverige bestämdes det att man skulle införa gregorianska kalendern succesivt mellan år 1700 och 1740 genom att ta bort skottdagarna var fjärde år vilket inte fungerade så bra i praktiken. Bland annat glömde man bort att man inte skulle ha skottår 1708.

2.3 (polymorfi, pekare till basklass, strömmar)
I den här uppgiften kommer du att skriva en klass `Calendar` som:

- internt håller en lista med händelser (såsom julafton, födelsedag) kopplade till datum
- har en defaultkonstruktor som sätter aktuellt datum till nuvarande datum.
- kan tilldelas en annan kalender (som kan vara instansierad med en annan datumtyp)
- har en metod `bool set_date(int år, månad, dag)` som sätter om aktuellt datum. Om parametrara bildar ett ogiltigt datum ska `false` returneras.
- har metoden `bool add_event` som kan ta en till fyra parametrar: händelse (string) och parametrarna dag, månad, år (int). Om någon av de senare parametrarna saknas så används aktuellt datum. Om datumet är ogiltigt ska `false` returneras. Om händelsen (strängen) redan finns i kalendern på det angivna datumet ska `false` returneras och ingenting nytt läggas in.
- `remove_event` med samma parametrar som `add_event`. Om det inte går att ta bort händelsen returneras `false`.

Använd STL för att hantera dina händelser. Använd `std::string` till strängarna som representerar händelser. En dålig lösning är att ha datum och händelse i två parallella vektorer med gemensamt index. Varför? Förklara dig vid redovisningstillfället.

Låt din kalenderklass vara en mallklass som tar en klass nedärvd från `Date` som argument. När du använder datum ska du endast använda dig av funktioner som ligger i basklassen `Date`. På så sätt håller du kalenderklassen datumoberoende. Några exempel på hur din klass ska bete sig:

```
std::cout << "----------------------------------------" << std::endl;
Calendar<Gregorian> cal;
cal.set_date(2000, 12, 2);
cal.add_event("Basketträning", 4, 12, 2000);
cal.add_event("Basketträning", 11, 12, 2000);
cal.add_event("Nyårsfrukost", 1, 1, 2001);
cal.add_event("Första advent", 1);          // år = 2000, månad = 12
cal.add_event("Vårdagjämning", 20, 3);      // år = 2000
cal.add_event("Julafton", 24, 12);
cal.add_event("Kalle Anka hälsar god jul", 24); // också på julafton
cal.add_event("Julafton", 24); // En likadan händelse samma datum ska
                               // ignoreras och inte sättas in i kalendern
cal.add_event("Min första cykel", 20, 12, 2000);
cal.remove_event("Basketträning", 4);

std::cout << cal; // OBS! Vårdagjämning och första advent är
                  // före nuvarande datum och skrivs inte ut
std::cout << "----------------------------------------" << std::endl;
cal.remove_event("Vårdagjämning", 20, 3, 2000);
cal.remove_event("Kalle Anka hälsar god jul", 24, 12, 2000);
```

```
cal.set_date(2000, 11, 2);
if (! cal.remove_event("Julafton", 24)) {
     std::cout << " cal.remove_event(\"Julafton\", 24) tar inte"<< std::endl
               << " bort något eftersom aktuell månad är november" << std::endl;
}
std::cout << "----------------------------------------" << std::endl;
std::cout << cal;
```

Man ska kunna skriva ut kalendern m.hj.a utskriftsoperatorn. Enbart händelser efter(!) aktuellt datum ska skrivas ut. Kalenderns händelser ska vara sorterade i första hand på datum och i andra hand på insättningsordning. Formatet ska vara på formen `datum : händelse` med efterföljande retur, se nedan. OBS! nollor framför ental (enligt Date) samt mellanslag före och efter kolon

```
----------------------------------------
2000-12-11 : Basketträning
2000-12-20 : Min första cykel
2000-12-24 : Julafton
2000-12-24 : Kalle Anka hälsar god jul
2001-01-01 : Nyårsfrukost
----------------------------------------
 cal.remove_event("Julafton", 24) tar inte
 bort något eftersom aktuell månad är november
----------------------------------------
2000-12-01 : Första advent
2000-12-11 : Basketträning
2000-12-20 : Min första cykel
2000-12-24 : Julafton
2001-01-01 : Nyårsfrukost
```

Testa din kod innan du skickar in den. Endast inskickad och testad kod kan redovisas. Redovisningen sker vid schemalagda labbtillfällen. Om redovisningen går bra och källkoden skickades in innan bonusdatum (står på labkvittot) får du två bonuspoäng. Det är mycket vanligt att man får komplettera sin labb innan den blir godkänd. Se för din egen skull till att få en underskrift på labbkvittot av handledaren oavsett om labben ska kompletteras eller inte.

Betygshöjande extrauppgifter

Extrauppgift 2.1 (6p) Lägg till följande funktionalitet hos kalendern:

- Skriv metoden `bool move_event(const Date & from, const Date & to, std::string event)` som kan flytta händelser i kalendern genom att först plocka ut och sedan lägga till händelsen på det nya datumet. Om händelsen inte finns i `from` alternativt redan finns i `to` returneras false.
- Skriv metoden `bool add_related_event(const Date & rel_date, int days, std::string rel_event, std::string new_event)`. En händelse ska kunna relateras till en annan händelse genom att man anger det antal dagar som skiljer dem åt. Flyttas händelsen kommer den relativa händelsen att flyttas också. Tas den bort så tas den relativa händelsen också bort.

- Ge möjlighet att lägga in återkommande händelser, t.ex. att julafton sker 24:e december varje år och fäktning är det varje fredag. Man ska kunna ange hur länge händelsen ska återkomma annars är defaultvärdet oändligt många gånger.
- Skriv en metod `print_events(const Date & start_date, const Date & end_date)` som skriver ut alla händelser från och med angett startdatum till och med slutdatum. Grunduppgiftens utskriftoperator ska skriva ut efter aktuellt datum fram till och med sista icke-återkommande händelse.
- Skriv en metod som tar bort alla återkommande händelser. Man ska kunna ta bort enskilda återkommande händelser (möten varje onsdag men mötet på onsdag när det är julafton är inställt).
- Ge möjlighet att lägga till födelsedagar så att kalendern räknar ut hur gamla födelsedagsbarnen är. I vår gregorianska kalender gäller att födelsedagar 29/2 fyller år 28/2 nästa år.
- Skriv ett demonstrationsprogram som visar den nya funktionaliteten. Skriv ut både instruktioner och utskrifter så att det går att följa utan att behöva slå upp i demonstrationsprogrammet vad som händer. Tänk på att testa fall långt fram i tiden.

Extrauppgift 2.2 (6p) Implementera ytterligare två utmatningsformat för din kalender. För att ändra format på utskriften använder man `cal.set_format(Calendar::format)`, där `Calendar::format` är en av tre `enum`typer, `list`, `cal` resv och `iCalendar` definierade i `Calendar`.

Det ena formatet liknar unix-programmet `cal`. December 2007 ska se ut så här med några speciella datum inlagda och aktuellt datum satt till 2/12.

```
       december 2007
må  ti  on  to  fr  lö  sö
                     1 < 2>
 3   4   5   6   7   8   9
10  11  12  13  14  15  16
17  18  19  20* 21  22  23
24* 25  26  27  28  29  30
31

 2000-12-20: Min andra cykel
 2000-12-24: Julafton
```

Det andra formatet är ICalendar (rfc2445) formatet som kan läsas in till andra kalenderprogram. Nedan är ett exempel. Ändra PRODID till någon unik text. Varje händelse är ett VEVENT med BEGIN/END. DSTART och DTEND innehåller datum och tider (efter T:et), sätt dem som entimmespass med början från klockan 8.SUMMARY innehåller en textbeskrivning över händelsen.

```
BEGIN:VCALENDAR
VERSION:2.0
PRODID:-//Simsalabim AB//En bra kalender made by ...//
```

```
BEGIN:VEVENT
DTSTART:20071220T080000
DTEND:20071220T090000
SUMMARY:Min andra cykel
END:VEVENT
BEGIN:VEVENT
DTSTART:20071224T080000
DTEND:20071224T090000
SUMMARY:Julafton
END:VEVENT
END:VCALENDAR
```

Redovisa genom att låta ett grafiskt kalenderprogram (t.ex. sunbird, google calendar m.m.) läsa in din kalenders ical-formaterade utskrift och visa upp vid redovisningen.