Övningar

Dag 4 – Introduktion till standardbibliotek

Vi kommer i dag att bygga vidare på vårt kalender-exempel. Vi kommer att öva mer på det vi gjort tidigare men även mer specifikt att kolla på dels nya funktioner på strängar och dels öva på standardbiblioteket.

En beskrivning av standardbiblioteket kan du finna på
`http://www.cplusplus.com/reference/stl/`

Övning 1

Öppna projektet Calendar där du skapade klasserna `Timestamp` och `Date`. som vi förra gången lade över i flera filer. Om du är osäker på om du gjort rätt, kolla på lösningsförslaget. Du kan även kopiera från lösningsförslaget.

I den gamla main-unktionen fanns bland annat följande rad.

```
cout << ts.getHours() << ":" << ts.getMinutes() << endl;
```

Hur ett objekt ska representeras som sträng är ett ansvar som oftast är bättre att överlämna till objektet. Det gör bl.a att representationen alltid blir lika i hela programmet. Vi ska därför skapa följande metoden string `toString()` i `Timestamp`.

Att konstruera en textsträng som är sammansatt av flera strängar är däremot lite knepigt. I gamla C var detta en mardröm. I C++-standardbibliotek finns en konstruktion som liknar utskrifterna. Det gör saken enklare.

Först måste fler bibliotek inkluderas, nämligen `<string>` och `<sstream>`.

```
#include <sstream>
#include <string>
#include <iostream>
using namespace std;
```

I stället för att skriva ut till konsolen, skrivs nu i stället till en sträng-buffert. I sista steget görs sedan bufferten om till en sträng. Funktionskroppen kan då se ut så här.

```
// Skapa en sträng-buffert
ostringstream os;
// Skriv till bufferen
os << getHours() << ": " <<  getMinutes();
// Skapa sträng från bufferten
string s = os.str();
// Returnera strängen
return s;
```

Observera att funktionerna `getHours()` och `getMinutes()` också är metoder på objektet. Därför behövs ingen variabel och punkt före anropet. I `main`-funktionen kan det nu stå

```
cout << ts.toString() << endl;
```

Övning 2

För att få inledande nollor, i t.ex. klockslaget `02:09`, behövs ytterligare funktionalitet för strömmarna. Detta finns i bl.a. paketet `<iomanip>`. Främst fyra funktioner är intressanta. För er som är bekant med formateringen i C känner igen funktionaliteten.

`setw(int)`	Bredden på fältet i antal tecken.
`setfill(char)`	Tecknet som ska fyllas upp med
`setprecision(int)`	Precision i antal decimaler som ska visas för flyttal
`setbase(int)`	Basen på heltal (ex. 2, 8, 10, 16).

För att få en inledande nolla vid ensiffriga siffror behöver vi dels sätta bredden på fältet till 2 och tecknet som ska fyllas ut med till '0'.

```
os << setw(2) << setfill('0') << getHours();
```

Justera `toString()` för `Timestamp` och skriv även motsvarande funktion för `Date`, så att utskrifterna blir på formen `2014-05-09`.

Övning 3

Om du vill kan du radera det som står i `main`-funktionen eller kommentera bort det. Du kan även döpa om den gamla funktionen och skapa en ny `main`-funktion även om det är lite ”fult”.

Testa nu att skapa en array med standardbibliotekets `vector`-klass. Skapa en array med kapaciteten 10 som kan lagra `Date`-objekt. Fyll nu arrayen med 10 objekt. Använd gärna `nextDate`-metoden för att få de 10 efterföljande datumen från i dag. Om du inte har metoden eller finner pekaren besvärlig, skapa objekten statiskt.

Låt objekten nu skrivas ut med en for-loop samt din `toString()` metod.

Övning 4

Ingen kod, men väl lite tankearbete. Antag att vi har en sekvens med 1000 heltal och att vi sedan vill

1) lägga till ett element först i sekvensen, samt
2) läsa element 500.

Hur många operationer krävs för respektive uppgift om vi har sekvensen som en

a) `array<int, 2000> // array med 2000 platser`

b) `vector<int>  // Säg att den rymmer 2000 element`

c) `list<int>  // Dubbellänkad lista`

För både `vector` och `list` finns en funktion som heter `insert`, men enbart `array` och `list` har `at` eller `[]`. `list` har dock iteratorer. Diskutera gärna med din granne, eller oss lärare.

Övning 5

Skapa två nya klasser, `Event` och `Calendar`. Båda dessa kan vara i samma fil, `calendar.h`, respektive `calendar.cpp`. Bygget av en egen liten kalender fortsätter. Dels tänker vi oss olika händelser (events), som består av ett datum, en tid, och en beskrivning. Dessa händelser läggs sedan till en lista som en namngiven kalender. Här blir det en del kodande. Förslag på klassbeskrivningar finns nedan, redo för h-filen.

```
class Event
{
private:
   Date date;
   Timestamp time;
   string descr;
public:
   Event(Date, Timestamp, string)
   toString();
};

class Calendar
{
private:
    list<Event> events;
    string name;
public:
    Calendar(string);
    addEvent(Event);
    clear();
    size();
    toString();
};
```

Skriv sedan en `main`-funktion där du först skapar ett `Calendar`-objekt och sedan lägger du in ett två eller tre `Event`-object. Skriv sedan ut händelserna med hjälp av `toString()`.

Tips

`Event::Event()`	Behöver enbart tilldela parametrarnas värden till objektets attribut (variabler)
`Event::toString()`	Använd `toString()`-metoderna på `Date` och `Timestamp`.
`Calendar::Calendar()`	Behöver enbart tilldela parametrarnas värden till objektets attribut (variabler)
`Calendar::addEvent()`	Anropa t.ex. `push_back()` på listan.
`Calendar::clear()` `Calendar::size()`	Anropa bara `clear()` och `size()` på listan.
`Calendar::toString()`	Loopa igenom listan med en `for`-loop och

anropa respektive elements `toString()`.

Övning 6 – överkurs

Testa att skriva om `Calendar`, så att det i stället är en lista av `Event*`, dvs pekare till objekt. Fundera speciellt hur `addEvent()`, och `clear()`-metoderna och en eventuell destruktor skulle se ut.