Laboration 6

Mål: Du ska på att konstruera och använda egna mallar (templates) i C++. Du ska också träna på användning av `struct`, `union`, bit-operatorer och bit-fält.

Läsanvisningar

Läs kap 14–15 i läroboken. Läs också föreläsningsbilder från de föreläsningar, som behandlar motsvarande avsnitt. Dessa finns på kursens hemsida.

Förberedelser

Läs igenom den inledande texten under rubriken ”Uppgifter” nedan och lös uppgift U2, U4, U5 och U7. Läs igenom och sätt dig in i uppgifterna U1, U3, U6 och U8. Uppgifterna U3-U4 är frivilliga.

Uppgifter

I denna uppgift ska du

U1. Skriv en funktionsmall, `skrivUtFalt()`, som har ett fält av godtycklig typ som parameter (fler parametrar kan krävas) och skriver ut fält på följande form:

```
[ 9, 7, 5, 3, 1 ]    // Fält av int
[ Kalle, Eva, Nisse] // Fält av string-objekt
```

Visa hur mallen används genom att skriva ut innehållet i fält av det slag som visas i exemplen ovan.
Vad krävs för att ett fält ska kunna skrivas ut med funktionsmallen?

U2. I lärobokens (C++ direkt) kapitel 5 (övning 18) beskrivs en effektiv sorteringsalgoritm som kallas *quicksort*. Utgå från algoritmen (eller lösningen till övningen) och skriv en funktionsmall (`quicksort.h`) som kan sortera godtyckliga datatyper med denna metod. Skriv också ett testprogram. Vilka egenskaper måste data som ska sorteras med funktionsmallen ha? Hur kan man komma förbi dessa?

U3. I C++ direkt sid 162 (3:e upplagan) finns en funktion som utför binärsökning i ett `int`-fält. Gör om funktionen till en funktionsmall och testa mallen med t ex ett `double`- och ett `int`-fält. Tänk dock på att binärsökning kräver att data är sorterade.

U4. I tidigare uppgift implementerades en stack med hjälp av ett fält. Gör om den till en klassmall. Skriv även ett testprogram som inför stackar med heltal och flyttal (värdena kan t ex slumpas fram).

U5. En tidigare uppgift gick ut på att implementera en kö med hjälp av ett fält. Gör om denna klass till en klassmall. Testa kön med framslumpade heltal.

U6. Skriv ett program som utnyttjar klassmallen i förra uppgiften för att hantera en kundkö där en kund representeras av en `struct`:

```
struct Kund {
  char namn[35];
  double belopp; // betalning
}
```

Kundernas (namn och belopp) läses in och placeras i kön. När en kund betjänas tas han eller hon bort från kön och får betala beloppet. Programmet ska hålla reda på hur många kunder som betjänats och totala försäljningssumman. Låt det hela ska styras via en meny där man får välja mellan att köa en ny kund, betjäna en kund eller avsluta. Antalet betjänade kunder och totalsumman redovisas löpande efter varje betjänad kund.

U7. Det finns många algoritmer för att komprimera data, dvs lagra data på ett mindre minneskrävande sätt. De flesta metoderna utnyttjar olika varianter av bitkodning.
Antag att vi vill behöver lagra datum. Ett naturligt sätt att representera datum är att utnyttja heltal (`int`): år, månad och dag, vilket kräver $3 * 4 = 12$ bytes om man utnyttjar vanliga (32 bitars) heltal för lagringen. Vi vill nu lagra detta på ett så minnessnålt som möjligt.
Vi kan enkelt halvera minnesåtgången genom att utnyttja 16 bitars heltal i stället för vanliga 32 bitars heltal. Fast minnesbehovet kan ytterligare reduceras om vi inser att dag och månad aldrig kan bli större än 31 resp 12, dvs 1 byte räcker för vardera (totalt $2+1+1 = 4$ bytes).
I själva verket kan vi med bitkodning få rum med alltihop i 2 bytes! För att inse detta gör vi följande observationer:

- För tal i intervallet 0..99 (årtalet) räcker 7 bitar (med 7 bitar klaras 0..127)
- För 1..12 (månadsnumret) räcker 4 bitar (klarar 0..15)
- För 1..31 (dagnumret) räcker 5 bitar (klarar 0..31)

Summa summarum: $(7+4+5) = 16$ bitar, vilket är två bytes, dvs $1/6$ av det ursprungliga minnesbehovet, en avsevärd besparing. Vi kan packa de tre datumkomponenterna i ett 2-bytes heltal enligt följande (Å=år, M=månad, D=dag):

Högre byte	Å	Å	Å	Å	Å	Å	Å	M
Lägre byte	M	M	M	D	D	D	D	D

a) Skriv en funktion, `packedDate` som returnerar ett packat datum enligt ovan. Dagnumret, månadsnumret och årtalet ska vara värdeparametrar.

b) Skriv en funktion, `unpackDate`, med ett packat datum som värdeparameter och med dag, månad och årtal i form av heltal som utparametrar.
Funktionen ska packa upp och leverera datum uppdelat i dess komponenter.

U8. Internt i datorn kan de i ett heltal ingående bytena lagras på olika sätt. Vanligast är rättvänd och bakvänd ordning (dvs med den mest signifikanta byten antingen först eller sist), men det finns även andra varianter,t ex kan bytena grupperas 2 och 2 och dessa 2- grupper lagras i rättvänd eller bakvänd ordning.

Rättvänd	4 bytes	12345678	2 bytes	1234	5678	1 bytes	12	34	56	78
Bakvänd	4 bytes	12345678	2 bytes	5678	1234	1 bytes	78	56	34	12

Att veta hur det ligger till kan vara viktigt, t ex när binära datafiler ska överföras från en dator till en annan. Ett sätt att undersöka hur det förhåller sig är att studera hur de olika delarna av ett värde lagras i minnet. Skriv ett program som i hexadecimal form skriver ut ett (4 bytes) heltal på tre olika sätt (som ett vanligt heltal, som två 2-bytes heltal och som fyra 1-bytes heltal). Utnyttja en `union` för ändamålet.