Laboration 2

Klass, operatoröverlagring, undantag

Denna uppgift innebär att konstruera en enskild klass Money för lagring av penningbelopp för valutor där hundradelar av valutaenheten används, till exempel kronor/ören och euro/centimes. Centrala delar i uppgiften är konstruktorer, operatoröverlagring, felhantering med undantag och namnrymder. Se kompletteringen **Operatoröverlagring**.

Tre tillfällen är schemalagda. Den introduktion till undantagshantering som ges på Fö 4 ska vara tillräcklig för det som krävs här, dvs att kunna definiera en egen undantagsklass med till exempel std::logic_error som grund och använda den för att signalera fel från Money-operationer (se föreläsningsmaterialet).

Följ planen enligt kursens eget schema på webben – eventuell komplettering görs på hemarbetstid!

Money-klassen ska definieras på en inkluderingsfil med namnet **Monetary.h**. Separatdefinierade medlemsfunktioner och annan kod som hör till Money-klassen men som ej behöver finnas i inkluderingsfilen ska definieras i filen **Monetary.cc**. Testprogrammet ska finnas på en tredje fil, **lab2.cc**. **Makefil** ska finnas.

Allmänt

Valutabeteckningen ska lagras som en sträng (std::string) till exempel "SEK" eller ”FFR”. Ett Money-objekt ska även kunna vara av ospecificerad valuta och valutabeteckningen ska i så fall vara tomma strängen, ””. Ett sådant objekt ska kunna fungera som ett generellt valutavärde och ingå i operationer med andra Money-objekt, oberoende av vilken valuta dessa andra objekt har.

Värdena för antalet enheter och antalet hundradelar ska lagras som två separata heltal. Negativa värden kan inte förekomma och det innebär fel om det uppstår i operationer.

Observera att det som ingår i uppgiften för G även ingår för VG. Man kan utan problem först kan lösa för G och sedan införa tilläggen som krävs för VG.

Felhantering

Fel ska hanteras med undantag. Härled en egen undantagsklass monetary_error från en lämplig klass i exception-hierarkin och kasta monetary_error-objekt med ett meddelande om felets art om fel uppstår i en operation. Följande fel ska kontrolleras, gärna fler om du kommer på sådana själv:

- Money-objekt får inte byta valuta. Money-objekt med ospecificerad valuta får dock tilldelas ett värde med angiven valuta och ska då anta den valutan
- valutor får inte blandas, till exempel försök att tilldela ett SEK-objekt ett FFR-objekt, att addera ett SEK-objekt och ett FFR-objekt eller att jämföra ett SEK-objekt och ett FFR-objekt – endast objekt med ospecificerad valuta får ingå i blandade operationer
- en operation som leder till ett negativt värde innebär ett fel, till exempel negativt initialvärde eller negativt värde som uppstår vid subtraktion eller nedstegning

Att värden som är för stora för att lagras kan uppstå, vid till exempel addition, behöver inte detekteras.

Tips

Tänk på följande när du konstruerar klassen `Money`:

- åtkomstspecifikation, dvs uppdelning av klassmedlemmarna mellan **public** och **private**
- användning av **const** då det gäller funktionsparametrar, returvärden och medlemsfunktioner
- typen för returvärdet från funktioner och operatorfunktioner (*lvalue*, *rvalue*)
- överlagrade operatorers semantik, t.ex. avseende argument och returvärde (*rvalue*, *lvalue*)

För G

Initiering ska kunna göras med ett strängvärde för valutan och heltal för enhetsvärde och hundradelar. Följande exempel visar de sätt som Money-objekt ska kunna initieras på:

```
Money m1;                 // Defaultinitiering, ospecificerad valuta: 0 enheter 0 hundradelar
Money m2{100);            // Ospecificerad valuta: 100 enheter 0 hundradelar
Money m3{10, 50);         // Ospecificerad valuta: 100 enheter 50 hundradelar
Money m4{"SEK");          // Svenska kronor: 0 kronor 0 öre
Money m5{"SEK", 10);      // Svenska kronor: 10 kronor, 0 öre
Money m6{"FFR", 100, 50);// Franska franc: 100 franc, 50 centimer
Money m7{m6);             // Franska franc: 100 franc, 50 centimer
Money m8{m1);             // Ospecificerad valuta: 0 enheter 0 hundradelar
```

En valutabeteckning ska antingen består av tre tecken eller vara ospecificerad, "". Det är också fel att ange ett enhetsvärde mindre än noll eller ett hundradelsvärde som är mindre än 0 eller större än 99.

Utskrift ska kunna göras på en utström en medlemsfunktion print() som tar en ostream som parameter. Om objektet som ska skrivas ut är av specificerad valuta ska valutabeteckningen skrivas ut först, sedan ett mellanrum och sedan beloppet med två decimaler, i annat fall skrivs enbart beloppet ut. Utskrift ska också kunna göras med **operator**<<. Exempel:

```
m5.print(cout);           // Utskrift: SEK 10.00 (utan efterföljande ny rad)

cout << m5 << endl;       // Utskrift: SEK 10.00
cout << m3 << endl;       // Utskrift: 100.50
```

Tilldelning av Money-objekt ska kunna göras med annat Money-objekt av samma valuta eller ospecificerad valuta. Ett Money-objekt av ospecificerad valuta ska även kunna tilldelas ett Money-objekt av specificerad valuta och ska då ändras till att vara av den specificerade valutan. Resultatet ska vara ett lvalue (kan t.ex. användas till vänster i en tilldelning) av typen Money. Exempel:

```
m4 = m5       // Okej, samma valuta
m4 = m3       // Okej, m3 är av ospecificerad valuta (m4 behåller aktuell valutaenhet)
m1 = m6       // Okej, m1, som var ospecificerad, erhåller FFR som valutaenhet, förutom beloppet
m4 = m6       // Fel, en specificerad valuta får ej ändras! (m4 är SEK, m6 är FFR)
```

Addition med + ska kunna göras med antingen två Money-objekt av samma valuta eller där det ena eller båda är av ospecificerad valuta. Två objekt av olika specificerade valutor ska inte kunna adderas. Resultatet ska vara ett *rvalue* (*right value*, dvs något som kan stå t.ex. till höger i en tilldelning, men inte till vänster) av typen Money. Exempel:

```
m4 + m5
m4 + m1
m1 + m2
m4 + m6         // Fel. olika valutor
```

Jämförelse mellan Money-objekt ska kunna göras med jämförelseoperatorerna < <=, > och >= och med likhetsoperatorerna == och !=. Implementera < och == ”direkt” och definiera de övriga i termer av dessa. Det ska inte vara tillåtet att jämföra två Money-objekt av olika specifika valutor; ett eller båda objekten kan vara av ospecificerad valuta. Några exempel:

```
m4 < m5
m4 != m3
m4 <= m6        // Fel, olika valutor
```

Stegning med ++ ska kunna göras och ska innebära att hundradelsvärdet stegas med 1. Både prefix och postfix ++ ska finnas. Resultatet av prefix ++ vara ett ändringsbart *lvalue*, medan resultatet av postfix ++ ska vara ett *rvalue* (*prvalue*). Exempel:

```
++m4            // Resultatet ska vara den uppdaterade operanden
m4++            // Uttryckets värde ska vara m4:s värde före stegningen
```

En medlemsfunktion currency() ska finnas, som ska returnerar valutaenheten i form av en string. Om objektet är av ospecificerad valuta ska en tom sträng returneras. Exempel:

```
string valuta{m4.currency()};
```

Namnrymd. Kapsla klassen Money i en namnrymd med namnet monetary.

Hjälpfunktioner och annat för att kunna implementera ovanstående funktioner och operatorfunktioner på ett bra sätt införs vid behov.

För VG

Utöver det som gäller för G, se ovan, ingår även följande.

Inläsning ska kunna göras från en inström med **operator**>>. Indata ska kunna ges både med och utan angivelse av valuta. Inmatning ska följa regeln att inget Money-objekt med specificerad valuta ska kunna byta valuta. Enhetsdelen och decimaldelen ska innehålla en siffra. Decimaltecknet ska vara punkt. Om valutaenhet anges ska den separeras från beloppsdelen med minst ett mellansrumstecken. Tillåtna inmatningsformat är:

```
SEK 10.50       // Valutabeteckning och belopp, åtskilda med mellanrum (ett eller flera)
10.50           // Enbart belopp, valutan ospecificerad, avslutande nolla är valfri
10              // Förenklat skrivsätt för beloppet 10.00
FFR 100         // 100 franska franc
USD 0.50        // 50 cent, nollan måste anges
```

Tänk noga igenom vad **operator**>> bör förutsätta om indata och hur den bör implementeras. Jämför med till exempel **operator**>> för **int**; denna

- läser först förbi eventuella vita tecken (motsvarar användning av manipulatorn ws på strömmen)
- läser sedan tecken så länge det kommer siffertecken och genererar ett motsvarande **int**-värde
- bryr sig inte om vad som följer efter det sista siffertecknet – mellanrum, radslut, filslut eller ett annat tecken än ett siffertecken – det får stå kvar i inströmmen till nästa läsning.
- om det första tecknet som kommer efter att inledande vita tecken har lästs bort inte är ett tecken som kan ingå i ett heltal (enligt C++), sätts *failbit* i strömmen och läsningen avbryts.

Det tål att tänka på hur man ska läsa till exempel SEK 10.50. En möjlighet är att se det som fem delar – tre bokstäver, ett eller flera mellanrumstecken, en eller flera siffror, punkt, en eller flera siffror.

Det ska kontrolleras om ett felaktigt värde ges vid inläsning med **operator**>>. Här kan en hel del kontrolleras eftersom det är text/tecken som läses in

Sammansatt tilldelning med addition, +=. När man har klasser där tilldelning (=) och addition (+) är tillåtet förväntas normalt även sammansatt tilldelning (+=) finnas. Samma möjligheter som angivits för addition ovan ska finnas. Resultatet ska vara ett lvalue av typen Money. Exempel:

```
m4 += m5        // Detsamma som m4 = m4 + m5
```

Stegning med `--` i analogi med ++, se G. Stegning får ej innebära att ett negativt belopp (-0.01) uppstår.

Subtraktion med -, i analogi med addition se G. Subtraktion får ej innebära att ett negativt belopp uppstår som resultat.

Sammansatt tilldelning med subtraktion, -=, analogi med sammansatt tilldelning med addition se G. Sammansatt tilldelning får ej innebära att ett negativt belopp uppstår som resultat.