Tekniska Högskolan i Linköping
Institutionen för Datavetenskap (IDA)
Torbjörn Jonsson 2005-01-09

Hantering av poster och enkla paket

I denna laboration kommer du att skapa ett paket med rutiner som kan hantera ett datum. För lagring av datum används datatypen post.

Mål

Du skall efter denna laboration kunna hantera poster samt skapa egna paket med rutiner som senare kan användas i ett större program. Undantag skall kunna hanteras både i och utanför paketet.

Uppgift

Att lagra ett datum går att göra på många olika sätt. T.ex. kan man lagra datumet som en textsträng ”1997-01-04” eller som tre heltal. I denna uppgift skall du lagra datumet i en post i vilken år, månad och dag lagras som tre heltal. Uppgiften är uppdelad för att du ska undvika alltför många saker på en gång. Man bör alltid försöka dela upp stora problem i mindre. Skriv ett litet testprogram som använder dina rutiner allt eftersom du programmerar.

Del A:

Skapa den datatyp som skall representera ett datum enligt beskrivningen ovan. Skriv dessutom de två procedurerna `Get` och `Put` som läser in ett datum från tangentbordet respektive skriver ut ett datum på skärmen. Ett exempel på kod som skall läsa in ett datum:

```
begin
  Ada.Text_IO.Put(”Ange ett datum: ”);
  Get(Date);  -- Detta är anropet till din ”Get”.
end ...;
```

Inläsningen skall då se ut enligt följande (observera att inläsningen kommer att kräva en sträng för inmatningen och att du sen konverterar innehållet till tre heltal som lagras i posten):

```
Ange ett datum: 1997-01-04
```

Utskriften av ett datum skall följa svensk standard, d.v.s. på formen ÅÅÅÅ-MM-DD. Ett exempel på på kod som skall skriva ut ett datum:

```
begin
  Ada.Text_IO.Put(”Ett datum: ”);
  Put(Date);  -- Detta är anropet till din ”Put”.
end ...;
```

Utskriften skall då bli:

```
Ett datum: 1997-01-04
```

Observera att du skall skriva procedurerna Get och Put och att dessa får innehålla anrop till de redan färdigdefinerade Get och Put som finns i Ada.Text_IO. Ingen felhantering med undantag (exceptions) i denna del. Detta kommer i först i del F.

Del B:

Skriv funktionerna `Next_Date` och `Previous_Date` som får ett datum (t.ex. 1997-01-04) som indata och returnerar nästa (1997-01-05) respektive föregående (1997-01-03) datum. Tänk på att det är olika antal dagar i olika månader.

Du behöver inte ta hänsyn till skottår, men om du vill det så är det skottår var fjärde år. Årtalet skall vara jämnt delbart med 4 om det är skottår. Dock finns det några undantag. Det är inte skottår då årtalet är jämnt delbart med 100. Dock är det skottår om årtalet är jämnt delbart med 400.

Del C:

De olika procedurerna som du nu skapat skulle kunna användas i ett antal olika program och för att man på ett enkelt sätt skall kunna återanvända dessa skapar vi ett *paket* av alltsammans. När du skapar ditt paket skall du inte ta med huvudprogrammet utan bara de procedurer och funktioner som skall återanvändas senare i andra program. I denna laboration är det alltså `Get`, `Put`, `Next_Date` och `Previous_Date` som skall finnas med i paketet samt hjälpprocedurer och hjälpfunktioner.

Ditt testprogram skall nu inkludera ditt paket och använda sig av de rutiner (procedurer och funktioner) som finns definierade där. Programmet skall givetvis fungera som tidigare.

Del D:

Skriv (i ditt paket) funktionerna `Less_Date`, `Equal_Date` och `Greater_Date` som jämför två datum (t.ex. 1997-04-04 och 1998-01-05) returnerar sant värde om första datumet är mindre, lika med respektive större än det andra datumet. De två datumen skall vara parametrar till funktionerna.

Modifiera dina funktioner så att de heter ”<”, ”=” respektive ”>”. Det du gör nu är att du skapar tre operatorer (du gör en operatoröverlagring) <, = och > som kan användas på samma sätt som motsvarande operatorer för heltal eller flyttal, men i detta fall gäller de för värden av typen datum.

Del E:

Skriv ett litet separat program som använder ditt paket. Ditt program skall till att börja med läsa in 10 datum i ett fält. Därefter skall programmet sortera dessa datum så att det tidigaste (minsta) datumet kommer först och till sist skall ditt program skriva ut datumen sorterade.

Del F:

Du skall nu ändra din procedur `Get` så att den även kan hantera inläsning av felaktiga data. Några exempel på felaktiga data är 2003-02-29, AAAA-BB-CC. Du kan säkert hitta på fler testfall (skall göras i del G senare). Din uppgift är att se till att `Get` upptäcker att det är något fel på datumet som läses in och ”resa/kasta” ett undantag (eng. ”exception”) som är namngivet relevant för felet. Du skall själv namnge undantagen i ditt paket (exempel på namn på undantag är t.ex. `Day_Error` och `Format_Error` för de två fall som finns givna ovan).

Observera att `Get` INTE skall fånga dessa undatag utan att den som anropar skall ta hand om felet och även utföra de åtgärder som behövs för att ”rätta till” det som går. `Get` skall alltså bara kasta undantaget. I denna del skall programmet alltså ”krascha” när undantaget kastas.

Man kan tänka sig två varianter på hur `Get` beter sig med tangentbordsbufferten. Det ena är att det första tecken som gör att det är fel (samt resten av inmatningen) lämnas kvar i bufferten och det andra är att fler tecken än nödvändigt har lästs in (och därför inte finns kvar i bufferten). Du avgör vilket som är bäst och motiverar ditt val. Det skall även stå med i modulbeskrivningen för `Get`.

Del G:

Skriv ett nytt program som använder ditt paket. Detta program skall testa om din `Get` fungerar som den skall.

Ett exempel på program som testar Get kan fungera på nedanstående sätt. Fråga din assistent vilket krav som finns i just din kurs.

Användaren kommer att mata in data enligt följande. Först tre tecken (”`OK `” eller ”`NO `”) som anger om det som följer är en korrekt inmatning eller ej. Till sist det som `Get` skall ”läsa in” (om de tre första tecknen är ”`OK `” är det ett korrekt datum som följer, är det ”`NO `” är det ett felaktigt data som matats in). Om det är ett ”NO ” skall nästa rad i inmatningen vara namnet på det undantag som skall kastas. Den sista raden som matas in skall vara ”`END`” och den anger att programmet skall avslutas. Antag att följande rader matas in (lägg dem i filen INDATA.TXT och starta ditt program enligt ”programnamn < INDATA.TXT” så slipper du mata in dessa varje gång):

Det finns två varianter att testa ditt program. Dels med korrekta indata och dels med felaktiga indata. I exempel 1 tar vi fallet med korrekta indata och i exempel 2 tar vi felaktiga indata.

`Exempel 1` på indatafil:

```
OK 2003-09-23
NO 2003-02-29
Day_Error
NO AAAA-BB-CC
Format_Error
END
```

Ditt program skall alltså läsa ett antal rader med testdata. För varje test som verkar vara fel (”`OK `” följs av att `Get` kastar ett undantag eller ”`NO `” inte resulterar i ett undantag) skall programmet skriva ut vilket radnummer testdatat hade, vilket förväntat resultat det var samt vilket fel det var.

Fall 1: Om din `Get` fungerar som den skall och de testdata du matat in (i filen INDATA.TXT) skall ditt testprogram inte skriva ut någonting på skärmen. Med ovanstående indata skall alltså programmet inte generera några utskrifter alls.

Fall 2: Antag däremot att din `Get` inte fungerar som den skall och att den för första datat kastar ett undantag (låt säga `Day_Error`) och i de övriga fallen inte kastar några undantag (maximalt fel alltså). Då skall utmatningen från ditt program bli något i stil med:

```
Rad 1: Borde varit OK, men fick ”Day_Error”.
Rad 2: Borde varit NO, men fick inget undantag.
Rad 4: Borde varit NO, men fick inget undantag.
```

Om din Get klarar av att kasta ett undantag skall programmet kontrollera att det är rätt undantag (står på raden efter det felaktigt inmatade ”datumet”) och kontrollera att det var ok. Antag att din Get gör rätt för inmatningarna på rad 1 och 4, men kastar Format_Error (istället för Day_Error) för inmatningen på rad 2. Då skall utmatningen från ditt program bli något i stil med (observera att endast ett felmeddelande skrivs ut):

```
Rad 2: Är NO, men fick ”Format_Error” istället för ”Day_Error”.
```

`Exempel 2` på indatafil:

```
OK 2003-02-29
NO 2003-02-28
Day_Error
OK AAAA-BB-CC
NO 2003-02-29
Format_Error
END
```

Fall 3: Om din `Get` fungerar som den skall och de testdata du matat in (i filen INDATA.TXT) skall ditt testprogram skriva ut ett antal felmeddelanden på skärmen. Med ovanstående indata skall programmet generera följande utskrifter.

```
Rad 1: Borde varit OK, men fick ”Day_Error”.
Rad 2: Borde varit NO, men fick inget undantag.
Rad 4: Borde varit OK, men fick ”Format_Error”.
Rad 5: Är NO, men fick ”Day_Error” istället för ”Format_Error”.
```

Det finns fler testfall att hitta på. Det är din uppgift att se till att du har tillräckligt många testfall så att det syns att du förstått hur man testar sina underprogram.

Del H: (frivillig)

Antag att vi har en variabel `Date` som innehåller ett datum och en annan variabel `Diff` som anger hur många år, månader och dagar framåt i tiden vi vill komma (både `Date` och `Diff` kan vara av datatypen datum). Det vore då praktiskt att kunna utföra operationen ”New_Date := Date + Diff;”.

I Ada kan man överlagra funktioner och procedurer och man kan t.ex. överlagra operatorn ”+” för olika datatyper. Detta ger oss möjlighet att utföra operationen ovan. Skriv funktionen ”+” som tar två argument (Date och Diff) och som ger ett resultat som motsvarar ovanstående.

Det kan vara praktiskt att skriva några hjälpfunktioner som utför addera år, månader respektive dagar för att klara av uppgiften. Funktionen ”+” kan ju sen använda sig av dessa för att utföra sin uppgift.

OBS! Gräv inte ner dig för djupt i vad som händer när man adderar en eller flera månader till ett datum. Du kan anta att det är 30 dagar som skall adderas för att addera en månad ävan om detta ger ett fel vid månader som inte är 30 dagar.