TDDE54/TDIU08/725G92/9AMA73/TDDD87: Ett sista tips om Ada.P2 ...

Torbjörn Jonsson <torbjorn.jonsson@liu.se>

Wed 15/12/2021 21:09

To: **TDDD87_2021HT_FP <tddd87_2021ht_fp@student.liu.se>; TDDE54_2021HT_AU <tdde54_2021ht_au@student.liu.se>; TDIU08_2021HT_C9 <tdiu08_2021ht_c9@student.liu.se>; 725G92_2021HT_Z3 <725g92_2021ht_z3@student.liu.se>**

Hejsan.

Först väldigt kort om O-uppgifterna sen direkt på det väsentliga om Ada.P2.

> Det stod att de uppgifter som skickas in på fredag (en 17/12) rättas på måndag och att vi efter det tar juluppehåll i det tidigare mailet om extra pass. En liten modifiering. Vi rättar förstås även det som inkommer i helgen (fram till den 19/12 alltså). Efter det är det marginell rättning av O-uppgifter fram till tentaperioden och endast 4 rättningspass i tentaperioden. Det går alltså att skicka in uppgifter under jul, men de rättas inte direkt.

Detta mail är ett sista tips för er som har Ada.P2 kvar att göra. Ni andra får ha en trevlig kväll och kan skippa resten (*om ni inte vill läsa förstås då det kanske i detta mail finns lite kuriosa även för er som programmerat tidigare eller klarat Ada.P2*).

Som ni kanske märkt kan det vara lite besvärligt att komma fram till vilken deluppgift i denna som motsvarar vilken typ av underprogram (det är lite därför det är värt betyg 3 i kursen). Jag tänkte ge ett par exempel där ni kan se lite hur man kan tänka. Jag använder denna veckas uppgift.

```
Deluppgift 1:
-------------
Skapa ett underprogram som räknar ut och returnerar
differensen mellan
ett heltal och ett flyttal. Underprogrammet skall ta emot
talen via
parameterlistan.

OBS! Inget skall skrivas ut inne i underprogrammet.

Deluppgift 2:
-------------
Skapa ett underprogram som har en heltalsparameter.
Underprogrammet
skall öka parameterns värde med 1 så att huvudprogrammet kan
skriva
ut det nya värdet.

Deluppgift 3:
-------------
Skriv ett underprogram som via parameterlistan får en sträng
S och ett
tecken C. Underprogrammet skall se till att huvudprogrammet
får
tillbaka en sträng som innehåller S:s värde där första
positionen är
utbytt mot värdet som C har.
```

Deluppgift 1:

När vi tittar på denna ser vi att den första deluppgiften handlar om att göra något som är en subtraktion av två data. Det spelar ju egentligen inte så stor roll om det är två heltal eller två flyttal eller om det hade varit två komplexa tal (d.v.s. två tal av typen "A+Bi", med en real- och en imaginärdel). En subtraktion av två heltal (t.ex. 27 och 18) hade vi ju i ett huvudprogram skrivit

som:

```
X := 27 - 18;     -- (1)
```

Detta gör att vi anropar en "operator" som heter "-" som i detta fall hanterar två heltal. I Ada kan man skriva detta som:

```
X := "-"(27, 18);     -- (2)
```

Om man tittar på den senare varianten (2) här så ser vi direkt att det ser "onaturligt" ut. Den första varianten (1) är ju så man skriver i matematiken och det är därför inte så att man skall använda variant (2). Att en sak går att göra (för att det ibland inte går att göra på annat sätt) betyder alltså inte att det är så man skall göra generellt sett. Jag antar att ni är med på detta.

Om vi byter ut ett av heltalen (i första deluppgiften alltså det andra talet) så måste vi helt plötsligt skapa en ny operator "-" då denna inte finns i språket. Vi skapar då en operator som skall klara av följande anrop från huvudprogrammet:

```
X := 27 - 18.3;     -- (3)
```

Denna operator skall då ha två parametrar (båda med "in" som parametermod) och den första skall ha datatypen Integer och den andra Float. OBS! Denna operator går inte att anropa på följande sätt:

```
X := 27.3 - 18;     -- (4)
```

I detta fall måste vi skapa ytterligare en operator "-" med omvända typer på parametrarna.

I uppgiften står det ofta som det gör i deluppgift 1 att man har två parametrar där den ena är heltal och den andra flyttal. Tanken är i 99% av fallen (som jag gissar alla som programmerat mycket känner likadant) att detta också anger ordningen av parametrarna. I uppgiften är vi alltså ute efter variant (3). *[Kommentar: Vi har varit "snälla" och givit ok även för variant (4) i vissa fall.]*

I detta fall var det ganska rakt på att det borde vara en operator som gör detta så vi får se om det krockar med någon av de andra deluppgifterna. Om det gör detta så får vi kanske tänka om och då kanske det blir en funktion eller kanske en procedur istället.

Undrar om det finns något som talar emot att det är en operator? Egentligen inte, MEN det står ingenstans att den skall "returnera" något (kan alltså vara en procedur lika gärna). Det är lite fingertoppskänsla som behövs här.

Deluppgift 2:

Denna är nog den deluppgift som är besvärligast här och det är nog lite halvsvårt även med deluppgift 3. Vi börjar med att fundera på om detta kanske är en operator. Kanske är detta en "+"-operator? Hur skulle den då anropas från huvudprogrammet?

```
X := "+"(5);     -- (5)
```

Detta skulle kunna fungera, men tillbaka till det onaturliga igen. Detta var ju inte så man gjorde även om det går. Dessutom skall resultatet bli talet 6. Hur får vi det att kännas naturligt? Några

kanske har hört talas om "++"-operatorn i t.ex. C++ och tycker att detta skulle vara en bra variant som motsvarar detta. I detta fall är det nog: Nja eller nej.

OBS! Denna, nummer (5), är egentligen den "unära" plus som finns (som går att skapa själv) som man normalt bara skriver:

```
X := +5;    -- (6)    Ger också resultatet 5 (d.v.s.
motsatsen till -5 som är negativt)
```

OBS! Detta gör direkt att vi INTE skall använda denna till att skapa något som gör att "+5" skall ge resultatet "6" som det ju skall vara i deluppgift 2. Vi kan nog utesluta att detta skall vara en operator då. YES!

Då går vi vidare och funderar på om det skall vara en funktion istället. Vi kan ju tänka oss t.ex. ett namn "Addera_Ett" och att denna funktion tar emot ett heltal och sen returnerar ett tal som är ett större. Skulle det se bra ut i ett huvudprogram?

```
X := Addera_Ett(5);    -- (7)    Ger resultatet 6, verkar väl
ok
```

Detta låter super eller hur? Vad talar emot detta? Jo, det står i deluppgiften att "Underprogrammet skall öka parameterns värde med 1". Vad betyder det? Det låter som att den parameter som underprogrammet har skall ändra värde tycker jag. Om det skulle vara så skulle alltså funktionen (7) ändra på talet 5 till att ha värdet 6. Låter ju lite absurt eller hur.

> *[Kommentar: I det gamla programspråket "Fortran 77" gick detta att göra. Väldigt farligt och ställde till det mycket när man skulle felsöka sina program senare då man inte förstår att 5 egentligen räknas som 6 om det är utfört enligt beskrivningen. ILLA alltså!]*

Tillbaka till detta. Just problemet med att parametern skall ÄNDRA VÄRDE gör att detta MÅSTE vara en procedur. Det är den enda varianten av underprogram i kursen som får göra detta.

> *[Kommentar: Jag vet att det i senare versioner av Ada går att ha "out" och "in out" i funktioner, men det är inte för det en bra modell att utgå från och vi jobbar med att ta fram bra modeller som man senare kanske behöver frångå, men ...]*

Jaha. Då blev det ju lätt. Om detta måste vara en procedur OCH parametern skall ändra värde behöver den ju absolut både ha "in" och "out" så det är ju samtidigt givet att huvudprogrammet skall skicka in värdet och sen skall underprogrammet "returnera"/"skicka tillbaka" det nya värdet via samma parameter. Detta gör också att man INTE kan anropa detta underprogram med konstanter!!!

Då kan det fortfarande vara rätt att deluppgift 1 är en operator. :-)

Deluppgift 3:

Nu kommer det som avgör allt! Kan detta vara en operator? Låter det rimligt? Hur ser vi det?

Vi kan tänka lite fritt nu och se om det kan se bra ut med en operator. Vi testar att anropa från huvudprogrammet med några vanliga operatorer och ser om det känns bra. Försök att känna vad det skulle betyda utan att veta om vad deluppgiften sa att det skulle ske där.

```
X := "="(Str, Char);    -- NEJ! Onaturligt på alla sätt!

X := Str = Char;    -- Hoppas att ni inte känner att detta
känns bra!

X := Str + Char;    -- Låter mer som att man sätter dit ett
tecken på slutet.

X := Str / Char;    -- Hur delar man en sträng med ett
tecken? Kanske går att tänka sig något.
```

Oavsett vilken operator vi väljer måste den stå i ett "uttryck", d.v.s. det som kommer tillbaka från operatorn måste användas till något. T.ex. i en tilldelning (som ovan) eller i ett villkor i en "if" eller ... Inget av dessa känns som att det kommer att leda till det vi vill (hoppas jag). Man skall inte försöka få fram en "krystad" variant. Då är det antagligen fel.

Vi säger att detta inte är en operator så är vi glada att vi valde rätt i deluppgift 1. YIPPIE!

Detta skall alltså vara en funktion. Det är det enda som är kvar denna gång. BRA!

Tänk nu på att en funktion tar emot data via parameterlistan (eller så har den ingen parameterlista om det inte finns "indata"). Funktioner "returnerar" data (ibland säger man "skickar tillbaka" vilket leder till strul som ni lätt inser). Det är bara ETT data som returneras, men detta data kan vara t.ex. en post med flera data inuti!

Sätt parametrarna i den ordning som det "sägs" eller "antyds" i uppgiften så blir det antagligen super. I detta fall alltså en sträng först och ett tecken som andra parameter. Returvärdet är av strängtyp. Anropet från huvudprogrammet kan vara likt detta:

```
Str_2 := Replace_First_Char(Str_1, Char);
Put(Str_2);
```

eller kanske direkt:

```
Put(Replace_First_Char(Str_1, Char));
```

OBS! Det skall INTE vara "out" eller "in out" i funktioner har vi redan pratat om. Det gör att man alltså INTE skall läsa in data från tangentbordet inne i denna funktion utan det görs i huvudprogrammet.

Det ni nu råkar ut för är: Ni kommer att få kompileringsfel om ni försöker ändra på den sträng som kommer in i er funktion då den är en "in"-parameter (konstant inne i funktionen!).

Vad göra? Vi vill returnera en ändrad variant av den sträng vi får in! Det går inte att ändra den! ARRRGH! Ända till man kommer på att underprogram får ha "lokala variabler" är detta kört. Sen är det trivialt.

Jag släpper resten av detta som hemövning och hoppas att ni nu har fått tillräckligt med nya tankar om hur man kan tänka.

Avslutning

Jag tror att vi nu kommit till slutet av denna session och jag hoppas att ni fått en liten större insikt i detta med underprogram och att det är en viktig värld. Därav vår noggrannhet i denna kurs gällande dessa och att vi vill att ni skall använda dessa för att dela upp problem. Det är i princip huvuddelen i allt som har med problemlösning att göra. "Divide and conquer" som är ett känt uttryck. Nyttja detta även i andra kurser, i livet och ni kommer att klara er bra. :-)

Hoppas att ni alla nu tar er igenom åtminstone Ada.P2 och att ni sen upptäcker hur "vackert" det kan vara när man förstår skönheten i detta med programmering. :-)

Ytterligare en gång: God Jul allihop!

M.v.h.
/TJ
--

- -
_/_/_/_/_/_/_/_/_/ ***Torbjörn Jonsson***
_/ _/ ***013-28 24 67***
_/ _/ ***Torbjorn.Jonsson@LiU.SE***
_/ _/ _/ *IDA/SaS/**UPP***
_/ _/_/ ***Institutionen för Datavetenskap***
- - - - - - - - - - ***Linköpings universitet***