TDDE54/TDDD87/TDIU08/725G92/9AMA73: Tips för Ada.O3.1 och O3.2 (och Ada.P3) ...

Torbjörn Jonsson <torbjorn.jonsson@liu.se>

Sun 26/09/2021 00:04

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 igen.

Nu har vi passerat det som handlar om "ren problemlösning" skulle jag vilja säga. Ni kan nu skapa stora program som löser alla problem (som är lösbara) med lite finess (med hjälp av underprogram). Nu kommer vi till den del som är super att ha för att kunna hantera stora datamängder och strukturera upp dem.

Nu är vi alltså inne på det som kommer lite senare i kursen och denna gång det som handlar om att man bygger datatstrukturer med en massa data i en och samma variabel. Detta mail handlar om både Ada.O3.1 och Ada.O3.2 då båda dessa hanterar delar av detta. Det finns en ytterligare del i detta (eller två beroende på hur man räknar), men dessa kommer som separata delar om de finns med i just din kurs. Isåfall får du ett mail senare om detta.

När det gäller datatstrukturerna i dessa två uppgifter så är de till viss del lika, men också till viss del olika förstås. Jag tar dem i ordning (och tar även med Ada.P3 i detta mail då den finns för några av kurserna).

Datastrukturen som vi kallar "fält" (Ada.O3.1)

Detta är egentligen en ganska stor sak och det vi tar om detta på FÖ visar att det egentligen är en generell datatyp som vi alltså kan skapa själva med egna uppsättningar data som passar vårt ändamål. Vi delar upp denna i två delar lite för att det blir mer praktiskt att prata om detta, men egentligen är det en enda struktur som är grunden och den kan vara flerdimensionell (d.v.s. det är inte begränsat till bara 2 eller 3 dimensioner utan kan vara 10-dimensionell eller mer om man vill).

Knepigt att tänka i något som är mer än tre dimensioner kanske. Vi struntar i detta och bara jobbar på så klarnar det oftast. :-)

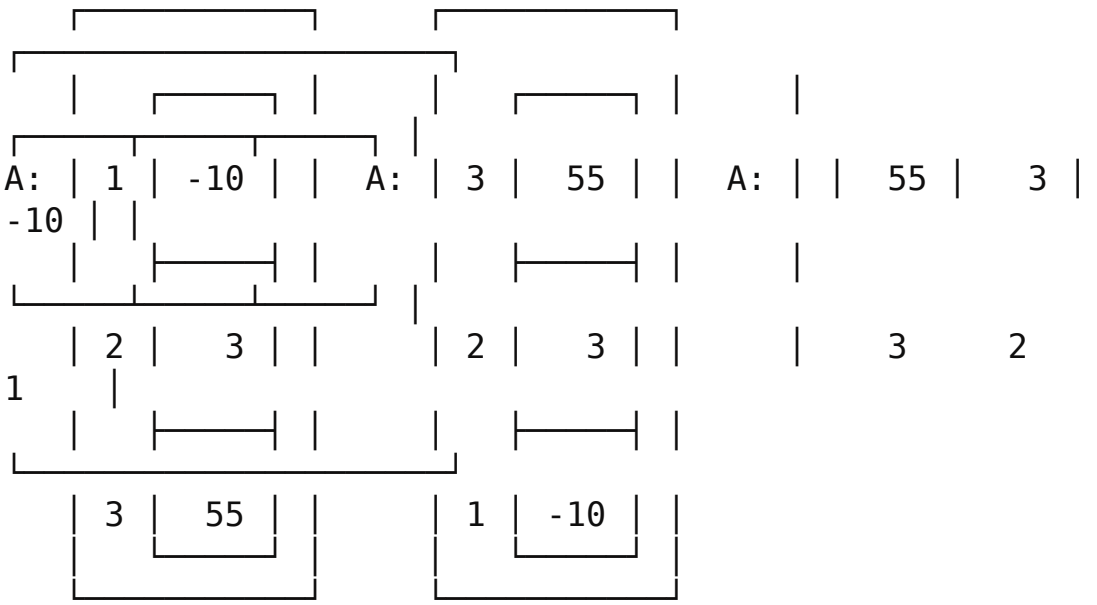
Det vi har dessa till är att lagra data. Samma sorts data genom hela strukturen. Vi kan sen kombinera denna struktur med andra eller likartade strukturer och på det viset skapa i princip oändligt stora datamängder (datorn begränsar oss med sin minneskapacitet dock).

Vad är det som är besvärligt med denna struktur kanske ni undrar. En hel del har det visat sig under åren jag haft undervisning för nybörjarstudenter, men egentligen är det inte svårt när man väl kommer på hur det hänger ihop. Några saker som dock skiljer sig från många (i princip alla andra) programspråk i detta är att Ada inte har begränsningen att man måste börja sina "index" på 1 eller 0. Detta är en fantastisk sak när man väl kommer på hur bra det är (det kanske passar bättre ihop med verkligheten man skall representera när man kan "leka runt" med sådant).

Innan vi kommer till detta med fördelarna skall vi se lite på vad som är typiskt för fält. Vi börjar

med en dimension. Det väsentliga är hur man vill strukturera sina data och vi utgår alltid ifrån en bild. Rita mycket i detta och du sparar tid och mycket energi.

En bild av ett **endimensionellt** fält kan se ut på massor av sätt. Ni har sett ett exempel på detta i uppgiften Ada.O3.1 (deluppgift 1). Jag ritar tre likartade med heltal istället för flyttal som data.



Du inser att jag skulle kunna flytta runt "indexen" (1, 2, 3) på ovansidan, högersidan, etc. så att det blir nya bilder. Du ser också att man kan tänka sig att räkna uppifrån och ner eller tvärsom. VIll inte fundera mer på antalet varianter på bilder som detta skulle kunna bli (och jag kan tänka mig en förenklad variant också som gör att det blir ännu knepigare.

Spelar det någon roll då hur man ritar bilden? Nej egentligen inte. Bara den visar det du har tänkt dig att lagra på ett adekvat sätt. Det är bilden som ligger till grund för hela programmeringen av detta.

Nu kommer det intressanta. Alla de bilderna som var ovan representeras på exakt ett sätt i ditt Ada-program. Det blir garanterat följande typdefinition av alla dessa:

```
type One_Dim_Array_Type is
  array (1 .. 3) of Integer;
```

Alltså: Du kan inte avgöra vilken bild du haft utifrån programkoden!

Viktigt att se är också att index 1 i fältet innehåller talet -10. I det deluppgifter vi har i Ada.O3.1 (och även Ada.O3.2) är det alltså viktigt att hålla reda på vilka index man har och var man skall placera de data som programmet skall läsa in. Fel på något av detta och det blir komplettering.

Vi har också sagt att det skall vara rätt identifierare på de ställen det det finns sådana. I detta fall skall VARIABELN heta A och fälttypen är alltså inte given till namn (denna får du bestämma i dessa uppgifter. Kanske inte så på andra uppgifter, men i just dessa är det så.

Om det i figurerna inte varit index 1, 2, 3 utan 5, 6, 7 eller -1, -2, -3 (detta är alltså intressant att det inte behöver gå från 1 eller 0 i Ada:s fält) skall alltså datatypen följa detta och det skall då vara "5 .. 7" eller "-3 .. -1" i definitionen av fälttypen (ja, det skall börja på -3 i sista fallet, fundera på varför).

Man kan även ha andra index än heltal (helt suveränt!) som t.ex. värden av typerna Boolean, Character eller egna påhittade uppräkningsbara datatyper (vissa kallar dem "enum"-typer, men det betyder egentligen bara att de skall vara UPPRÄKNINGSBARA). Funkar alltså inte med Float.

En liten strulig sak är att det i vissa delar finns saker som sätter käppar i hjulen (alltid en verklighet vi sitter i). T.ex. är det lite strul med Character. Det finns nämligen två datatyper till som håller på med tecken som ni kanske fått höra tidigare. Alla de som jag åsyftar har samma sätt attt beskriva hur man visar att det är ett konstant-tecken och det är genom att skriva apostrofer runt tecknet. T.ex. är 'a' en Character har ju vi sagt, men det skulle lika gärna kunna vara en Wide_Character eller en Wide_Wide_Character. Struligt. Javisst, men "vi struntar i det" och bara tar tanken till hur gör man för att hantera detta då.

För att definiera ett intervall (t.ex. 1 .. 3 i en for-sats eller i en fältdatatypsbeskrivning för indexen) kan an lägga till vilken datatyp man vill ha detta intervall. Vi skulle kunna skriva " Integer range 1 .. 3 " i definitionen ovan (men det behövs in för att Integer-konstanterna har inget likartat problem som Character-konstanterna *[eller egentligen kanske det finns detta, men att jag luras lite för att undvika de problem som då uppstår :-)]*. På samma sätt kan man specificera att det istället skall vara ett teckenintervall med " Character range 'a' .. 'c' ".

OBS! I de uppgifter vi har skrivs det ibland "True" och "False" i dessa datastrukturer och vi ser då dessa ALLTID som värden av typen Boolean.

Din uppgift är alltså att tänka till över vad som är lämpligt som "index" och vilket data som skall vara inuti fältet och vilken ordning index skall vara i definitionen av datatypen samt se till att de inmatningar och utskrifter som programmet hanterar lagrar och hämtar på rätt plats i fältet.

När vi nu kommer till **flerdimensionella** fält begränsar jag mig till matrisen (som är 2-dimensionell i vår kurs), men det skulle kunna vara fler dimensioner. Egentligen bara att hålla tungan rätt i munnen när man indexerar de olika dimensionerna.

Tips: Bra att tänka på hur man namnger saker när man kommer till mer komplexa problem och det är därför viktigt med "bra variabelnamn". I dessa uppgifter har vi genererat identifierarna lite med hjälp av slump för att kunna testa er på lite andra sätt, men tänk på variabelnamn när ni programmerar i övrigt.

Bilden som visar en matris är alltså den som har variabelnamnet DS3 i Ada.O3.1. Tänk även här på att bilderna kan roteras, vridas, speglas och lekas med på en massa sätt så det gäller att "komma på" hur det hänger ihop och vad som är vad.

När det gäller matriser är det en sak som skiljer sig från de endimensionella fälten. Man kan ha två olika implementationer (d.v.s. datatypen kan bli rätt på två sätt). Se följande två datatyper:

```
type Matrix_1_Type is
  array (1 .. 3, 1 .. 4) of Float;

type Matrix_2_Type is
  array (1 .. 4, 1 .. 3) of Float;
```

Dessa två kan man rita på samma sätt (på en massa olika sätt). D.v.s. det måste här även hjälpas till i programkoden att förklaras vad som är vad för att man skall arbeta ihop med andra

på ett bra sätt.

På samma sätt som innan är det egentligen bara i dessa uppgifter att "komma på" hur man refererar till ett visst data och sen är det upp till er att visa att ni kan få ihop samma sak som för det endimensionella fältet.

Man kan förstås kombinera fält med varandra också. Man kan t.ex. ha ett endimensionellt fält inne i varje datadel i en matris eller matriser inuti ett endimensionellt fält eller ... Ni ser vad vi har att leka med här och tänk vilka möjligheter vi har att rita figurer på olika sätt för att se om ni kan hantera och förstå hur ni skall skriva era datatyper. Ett exempel på detta är DS2 i uppgiften.

OBS! Det är lätt när man kommit på det och det är inte långt borta utan det kravs bara att ni ritar figurer och skapar typer och övar. Därav de tre delarna för att vi skall visa på åtminstone lite hur det ser ut och för att ni skall komma igång.

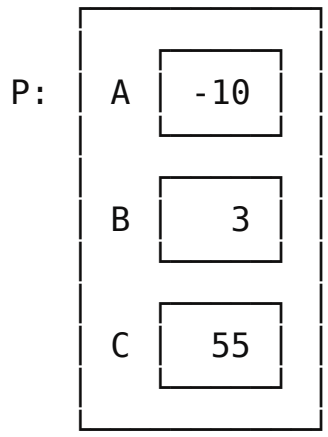
TIPS: Tänk på ordningen ni stoppar in data i datastrukturen. Det gör att ni kommer rätt snabbare förstås.

Denna uppgift är en liten rackare och den ställer till det lite för en del så räkna med att ni behöver både diskutera en del med assistenten och att ni kan råka ut för både automaträttningsfel och assistenträttningskompletteringar. Tag vara på chanserna att rätta till saker när det kommer tillbaka mail om denna alltså (gäller väl alltid, men ...).

TIPS: Den vanliga "for"-satsen är väldigt behändig ihop med fält. Ibland kan det finnas anledning att använda de andra "looparna", men tänk till om var och när så att du inte strular till det i onödan. Attributen till fält är också av godo. De kan verkligen hjälpa en att slippa leta fel i sina program. Nyttja dem!

Datastrukturen vi kallar "post" (Ada.O3.2)

Detta med "poster" är nog lite lättare på ett sätt än fält, men det gäller samtidigt att veta att de har många likheter. Vi skulle t.ex. kunna rita en post på samma sätt som vi ritar fält (och självklart vice versa), men vi försöker hålla oss lite renodlade i uppgifterna och som ni ser i denna bild sitter inte de tre "delvariablerna" A, B, C i postvariablen P ihop (rent bildmässigt). Detta är alltså inget som är ett måste utan det viktiga är egentligen vad som talar om vilket data man refererar och det ser man direkt skillnad på när man är mer van.



OBS! En stor skillnad är alltså att vi inte har "index" i en post utan det är ett gäng "variabler" som är "ihopklumpade" i en omslutande variabel. Vi brukar kalla de inre identifierarna för

"delvariabler" då det blir lite struligt om man direktöversätter det engelska ordet "field" till svenska (borde då bli "fält" och det inser ju alla att vi inte vill krocka med i denna beskrivning).

På samma sätt som för Ada.O3.1 gäller i Ada.O3.2 att ni skall skapa rätt datatyp, ha rätt variabelnamn och att alla delvariabler har rätt namn. Bilden talar inte heller i detta fall om vad datatypen skall heta, men det skulle kunna finnas med i specifikationen då det är av stor vikt att man som programmerare kan skicka sina data till dem som skriver de underprogram som vill använda datatyperna.

Tips här är att vi alltså inte använder "True" och "False" som delvariabelnamn i våra figurer.

Mer komplexa datastrukturer (Ada.P3)

Vissa av kurserna har som examinationsmoment för att få högre betyg Ada.P3 som möjlighet. Denna kommer att innefatta en datastruktur som är en kombination av ovanstående delar. Vi kommer att lägga ut några testvarianter som ni kan öva på i förväg och på samma sätt som för Ada.P2 kommer de sen att dyka upp på kurshemsidan under det "arkiv" av sådana uppgifter under Ada.P3-områdets hemsidedel.

När det gäller Ada.P3 kommer vi att ställa till det lite mer och kraven är förstås som vabligt att sakerna går via automaträttningen och sen via assistent. Då det är givet hur uppgifterna "i princip" ser ut är kraven alltså hårda på att det är rätt.

Saker som kommer att ställa till det lite är att det i vissa fall kan vara så att man får lite strul med de ordinarie paketen (t.ex. "Ada.Text_IO") genom att de datatyper man skapar "krockar" med strängar till vissa del. Här gäller det då att kunna lösa problemet med detta (man får kompileringsfel som man måste lösa).

Andra saker är att det blir mer komplext. Tänk dig en matris med poster som i sin tur innehåller fält. Kul grejor eller hur. Eller andra kombinationer som bara fantasin sätter gränserna för.

Nivån (både innehållsmässigt och rättningsmässigt) på Ada.P3 är alltså lite tuffare än Ada.P2 då det nu rör sig om nästa betygsnivå.

Allmänna tips som hjälper dig med datastrukturuppgifterna (och annat)

Här följer några tips för dem som kört fast eller behöver lite tips för att komma igång.

1. Underprogram är av godo!

2. Dela upp problemen i mindre delar. För många satser inuti varandra (t.ex. "loop" i "if" i "for" i ...) gör att det inte blir lätt att rätta till saker. Bryt ut delar och gör underprogram!

3. Tänk er en datatyp som något som kanske går att "bryta ut" som en enhet. Kanske går det att sköta denna separat och på det sättet minska komplexiteten. Kanske hålla ihop underprogram som hör ihop med en datatyp rent textmässigt i sin kod? *[Detta kan dock strida mot tidigare tips till en viss del, men kan i vissa fall vara bra. Försök dock hålla ordningen på konstanter, datatyper, underprogram och variabler så blir det lättare med allt så länge.]*

4. Ibland kan det vara lättare att inte bryta isär saker. Ajdå. Ett tips som går emot ett annat. Jo, det är detta som är själva problemlösningen i programmeringen. Att se flera olika sätt att lösa ett problem och välja det som verkar bäst.

5. RITA FIGURER!

6. Kolla noga vilka data som skall läsas in i körexemplet och se till att de hamnar i rätt "index" eller "delvariabel".

7. Inläsningen eller utskriften kanske är bakvänd. Specifikt i Ada.P3 kommer det att vara en del av problemet att komma på hur det hänger ihop.

8. När ni skall mata in värden som motsvarar "True" och "False" är det lite besvärligt. Finns ingen direkt "Get" för detta. Vi räknar med att ni istället får läsa in ett tecken 'T' eller 'F' och sen "omvandla" detta (fundera på hur, det är inte svårt) till en "Boolean".

9. Lös inte hela uppgiften på en gång! Lös en del och testa och sen går du vidare.

10. Underprogram är av godo! *[Ja, det är sant även andra gången jag säger det. :-)]*

11. Bästa tipset för idag är nog: RITA FIGURER! Går inte att säga för många gånger.

En viktig sak som alltid gäller: Skjut inte upp saker. Planera din tid. Det är lite av det man behöver som ny på universitetet (för er som redan har varit här är det bara en påminnelse. :-)). Planera in ledighet också. Det är viktigt med sömn, motion och frukost. Till sist också: Sprid glädje. Det gör alltid saker lättare. Även när det är som mest jobbigt skall man försöka se det positiva i det som händer. Då orkar man i mål.

Avslutning

Nu börjar jag låta som om jag är din förälder. Inte min mening. Vill bara ge er lite tips i all välmening. Dock har era föräldrar antagligen rätt i det mesta de sagt till er. Även om det inte är kul att erkänna. :-)

Har ni väl kommit till denna del i kursen börjar det roliga. Snart kan ni skapa vilka program som helst.

Ha nu en bra dag och njut av livet.

M.v.h.
/TJ
--

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