TDDE54/TDDD87/TDIU08/725G92/9AMA73: Tips för Ada.O2 (och Ada.P2) ...

Torbjörn Jonsson <torbjorn.jonsson@liu.se>

Fri 24/09/2021 12:18

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.

Denna uppgift är gjord för att ni skall börja tänka på när man skall använda vilken typ av underprogram. Det är också meningen att ni skall komma in i detta med att "dokumentera" vad som händer vid parameteröverföringen (d.v.s. skriva tydligt vad som är "in", "out" respektive "in out" samt vad som skall returneras) vid kommunikationen mellan huvudprogram och underprogram. Ytterligare en sak som är väsentlig är hur man använder data och var de skall vara "definerade/deklarerade" (det som vi kallar "räckvidd" eller på engelska "scope").

Underprogrammen är i dessa uppgifter inte stora, men det är viktigt att de gör det som står och inte varken mer eller mindre. Om man har undringar om detta frågar man bara (speciellt under P-passen är det viktigt att ställa frågor om man är tveksam). Under P-passen i SC och i övrigt frågar man förstås assistenten. Vi får ibland lite tvetydiga frågor som då kan ge svar som kanske kan misstolkas så försök ställa raka frågor. Skrift är svårt så vi hoppas att kommunikationen inte blir galen där, men det händer tyvärr.

Det finns i Ada tre typer av underprogram som är lite olika i karaktär. När man använder vilken av dessa beror till största delen på hur man vill anropa dem i huvudprogrammet. Sen finns problematiken med hur man skickar data.

I uppgifterna är det tänkt att det ska vara ett underprogram av varje sort. Om det går att välja mellan två varianter finns det säkert något som utesluter en variant via något av de andra programmen.

När skall jag använda vilket underprogram då?

Något som brukar sägas är att "allt" kan lösas med procedurer. En sanning med modifikation, men i princip är detta samma sak som att allt går att lösa med en "while" gällande upprepning. Går att göra, men väldigt osmidigt i många lägen.

Den första frågan man bör ställa sig är iallafall. Skall vi få tillbaka ETT värde från underprogrammet till huvudprogrammet? Om svaret är NEJ på den frågan är det absolut en procedur vi pratar om. Funktioner och operatorer måste nämligen returnera exakt ett värde.

Den andra frågan man kan ta direkt är om det är så att underprogrammet skall påverka någon eller några av sina parametrars värde (d.v.s. ha "out" eller "in out"). Detta säger direkt också att det bör vara en procedur. I vissa språk behöver man lösa saker via utparametrar i funktioner, men det beror till 99% av fallen på att man inte har begreppet procedur. I Ada "går" det att ha utdata via funktioner, MEN det är absolut en dålig idé om man arbetar i en större skala. Renheten man får av funktioner som gör på "rätt" sätt gör att man inte får knasiga fel senare i sina (stora) program. I denna kurs säger vi: "INTE utparametrar i funktioner". Det ger inte G på uppgiften.

En fråga som är av stor relevans är om det man vill göra något som liknar en normal beräkning ("+", "-", "*", "/" eller "**") eller någon jämförelse ("<", ">", "<=", ">=", "=" eller "/=") som finns för de normala enkla datatyperna, eller möjligen en sammanslagning av två data (som "&" för strängar). Har man två data som man vill göra något liknande med är det mycket troligt att man hamnar i att de är en operator man vill skapa. Viktigt här är att man när man anropar en operator i huvudprogrammet vill säga något i stil med "A + B" eller "A < B". Operatorn arbetar med sina operander (gamla saker från matematik i grundskolan hoppas jag) genom att ta deras värden och utföra något som ger ETT resultat.

Frågan om hur man vill anropa sitt underprogram i huvudprogrammet är av stor vikt. Tänk dig att du vill kunna göra något som "räknar ut" eller "tar fram ett värde" där du vill kunna anropa ditt underprogram om sen direkt anropa ett annat underprogram för en annan likartad beräkning. Då pratar vi om att man vill skapa "uttryck" och då blir det direkt tal om funktioner. Exempel på detta är "Sin", "Cos", "Factorial", etc. Alla dessa skulle vi kunna anropa i en sekvens och t.ex. sätta "+" mellan anropen (ex. "Z := Sin(X) + Cos(Y);"). I detta fall är det helt omöjligt att skapa detta som procedurer eftersom procedurer inte RETURNERAR ett värde som huvudprogrammet kan använda till uttrycket.

OBS! Vilken del i programmet skall göra vad? Generellt sett får man fundera på om det är "logiskt" att underprogrammet skall utföra en uppgift eller om det är med logiskt att huvudprogrammet gör detta. Tänk då på att huvudprogrammet inte skall få tillbaka saker från underprogrammet som det egentligen inte behöver och det skall heller inte behövs skicka in saker till underprogrammet för att detta skall fungera om det inte är data som verkligen inte underprogrammet kan sköta själv.

Någon kanske säger nu "Du sa ju att det alltid gick att göra procedurer!" och det är fortfarande rätt. Jag skulle kunna skriva procedurer som gör samma sak, men det blir "osmidigt" att göra sakerna i huvudprogrammet. Jag visar ett exempel på hur det skulle kunna se ut.

Med funktion:

```
    Z := Sin(X);    -- Z får resultatet via returvärdet från
funktionen.
```

Med procedur:

```
    Sin(X, Z);    -- Z blir en extra parameter som
"returnerar" svaret.
```

Det komplexare uttrycket ovan skulle bli "fult". Vi ser på det också:

Med procedurer (ett exempel av flera på hur man kan göra):

```
    Sin(X, Z1);
    Cos(Y, Z2);
    Z := Z1 + Z2;
```

Det gick alltså att lösa, men det blir dels mer komplext och dels "fulare" i någon bemärkelse. Man kommer fortare fram till bra lösningar om man kan skriva "fin kod" och att man nyttjar det som är "bra varianter". Det är detta vi jobbar med i denna kurs. Om ni tittar och hittar saker på nätet kommer ni att kunna se att det ofta är lösningar som är "fula" och kanske rent av "dåliga" när ni förstått detta.

Det finns mycket man kan prata om hur man väljer typ av underprogram, men jag stannar här

och låter er diskutera med varandra och komma fram till bra val.

Det här med "räckvidd" då? Vad är detta?

Här kommer vi in på det som handlar om VAR man definierar/deklarerar sina variabler, typer, konstanter i sitt program. Det gäller också var man deklarerar sina underprogram (och paket senare).

Om man följer mallen att variabler skall ligga precis före "begin" kommer man alltid att hamna rätt vad det gäller hur man bör (läs "skall") göra med användning av "globala variabler". Att lägga variabler ovanför underprogram innebär att du kan skapa problem för dig själv om du råkar använda dessa i dina underprogram. I denna kurs säger vi: "Inte tillåtet att använda globala variabler i underprogram" och det betyder att alla variabler du använder i ditt underprogram skall antingen vara lokalt deklarerade före "begin" i underprogrammet eller som parametrar i underprogrammet.

Konstanter är något som vissa blandar ihop med variabler. OBS! Skriver man "constant" när man definierar sin "variabel" är det inte en variabel utan en konstant. Dessa kan man (läs "bör man") definiera först i sitt program (eller möjligen först i sitt underprogram om man har dem lokalt). Konstanter är till för hela programmet för att man skall slippa få strul med olika värden på samma sak.

Datatyper (slarvigt kallat "typer") behöver ofta konstanter och används av underprogram och när man skapar variabler. Behöver alltså ligger ovanför dessa (d.v.s. grundregeln blir att lägga dem direkt under konstanterna).

Underprogrammen. Lägg dem i rätt ordning (vi har inte i denna kurs mer än tagit upp att det går att skapa underprogram på separata filer, men det är inte så man brukar arbeta i det stora så lägg dem inuti huvudprogrammet (eller i "paket" senare). Anrop till ett underprogram skall ske nedanför definitionen (eller deklarationen) av underprogrammet.

Jag stannar där för att inte gör en bok av detta också. OBS! Detta område är stort. Det finns massor av saker som man borde kunna, men vi begränsar oss och tar det mest väsentliga i kursen.

Begreppet "moder" och varför det är viktigt

Vi är lite petiga även med detta som har med "moder" att göra i kursen som jag skrev ovan. Varför? Jo, det är så att när man skriver program skall man inte göra det för att lösa ett specifikt problem och bara för sig själv. Det handlar om att man arbetar i stora lag av programmerare i många fall. Det gör att man skall vara tydlig med hur man skriver sin kod. Här ingår att man skall beskriva så bra det går hur man t.ex. kommunicerar mellan de olika enheterna i programmet (i detta fall underprogrammets gränssnitt mot den som anropar det).

Att INTE skriva någon mod i parameterlistan betyder i Ada att man har moden "in", men är detta tydligt för den som läser detta och är det verkligen så att du har tänkt till om hur du egentligen ville göra? Troligen inte i båda fallen. När man "slarvar" med sådant förlorar man oftast mycket tid i felsökning senare om man inte gjort rätt. Det vill vi att ni slipper.

Att ha "attityden" (förlåt ordvalet, men ...) "att det spelar väl ingen roll, det blir ju rätt" gör tyvärr att många program kostar mycket mer än de borde i tid att skapa och rätta till. Tänk om där så blir du (även om du redan är en duktig programmerare) superbra!

Ada.P2-uppgifterna då?

I Ada.P2 kommer vi att vara väldigt petiga med att ni väljer bra och att ni använder det som har med underprogram att göra på rätt sätt. Uppgifterna är mindre än vanliga tentauppgifter och vi har därför satt upp kravet till att det skall fungera (rätt saker i underprogrammen) och att allt gällande "moder" och val av underprogram är ok. Det gäller i detta fall betyg 3 eller G för motsvarande tenta.

Några extra tips på vägen

För att höja tempot lite för er och ge er en möjlighet att komma i mål så vill jag ge några enkla tips som det är meningen att man skall komma in i under denna kurs. Jag hoppas att ni förstår att dessa är till för er och att de endast är till för att ni skall tjäna på det.

1. Var noggrann. Jag har sagt det förut, men det är superviktigt för att man skall slippa en massa strul. I ert fall tar det ju bara tid att vänta på kompletteringar och det vill varken ni eller vi att ni skall behöva göra.

2. Se inte småfel som ett hinder. Om ni får tillbaka ett fel som är ett "stavfel" eller liknande. Bara fixa det och skicka in igen. Det är därför vi skickar ut kompletteringar samma dag (flera gånger). Dessa fel försvinner mer och mer om man är noggrann, men det finns alltid saker som man missat så ...

3. Ni behöver komma upp till nivån att ni kan felsöka era egna program i kursen. Detta innebär att ni måste kunna hitta på egna testfall som överensstämmer med det som står i uppgiften. Se uppgiften som en "specifikation". För att öva på detta kan man t.ex. "ge sig på" kompisens program och försöka "knäcka" det med testfall som gör att dennes program inte fungerar. Gör ni det mot varandra så kommer ni att hitta era egna fel också.

4. Snart kommer det att finnas många "gula" uppgifter att göra. Vänta inte med att skicka in nästa uppgift bara för att du inte fått godkänt på den förra än. Vi räknar med att ni arbetar på nästa uppgift när ni inte har något att göra med den som ni skickat in. Även de som är "grå" går att jobba vidare på som ni vet. Tanken är att man ligger "ett steg före" även där om man inte har något att göra i övrigt.

Jag hoppas att ni nu är på gång och att ni börjar komma igång med hur allt fungerar. Det kommer vad jag gissar att "explodera" med inskickningar när ni kommer igång ordentligt.

Avslutning

Efter Ada.O2 släpps som sagt en mängd uppgifter upp så nu börjar det lite mer roliga i kursen. Vi kommer fortfarande att ha automaträttningen på dessa, men vi kommer att vara fokuserade på det som har med området att göra i rättningarna så det kan nog kännas som att vi är lite snällare när vi kommer framöver nu.

Lycka nu till med Ada.O2 och Ada.P2 och alla andra uppgifter framöver.

M.v.h.
/TJ
--

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