Tekniska Högskolan i Linköping
Institutionen för Datavetenskap (IDA)
Torbjörn Jonsson 2011-09-07

Fälthantering med kommandoradshantering

I denna laboration kommer vi hantera ett fält (en array) fylld med heltal på lite olika sätt. Vi kommer att behandla endimesionella och även tvådimensionella fält. Dessutom skall du ha provat på att starta ditt program med indata från kommandoraden.

Mål

Studenten skall efter denna laboration kunna hantera fältstrukturer, iterera över dessa utan risk för referens till element utanför fältet. Dessutom skall studenten förstå hur kommandoradsargument fungerar och kunna ta in dessa i sitt program.

Uppgift

Uppgifterna som följer kommer att vara varianter på hur man sorterar data. Det spelar inte någon roll vilken sorteringsalgoritm ni använder.

OBS! Det är endast tillåtet att i underprogram använda variabler som skickas som parametrar till eller från underprogrammen samt lokala variabler, d.v.s. det är inte tillåtet att använda sig av globala variabler inuti underprogrammen. Används globala variabler fås laborationen i retur utan åtgärd.

Del A:

Du skall skriva ett program som genererar 20 heltal (som skall ligga i intervallet [1, 10]) och lagrar dem i ett endimensionellt fält. Därefter skall talen skrivas ut på skärmen (osorterade), sorteras i stigande ordning (det minsta talet först i fältet) och till sist skrivas ut på skärmen (ett tal per rad).

För att underlätta framtida ändringar är det nödvändigt att dela upp programmet i underprogram. Lämpliga underprogram (för denna deluppgift) är `Generate` (en parameter, fältet med talen), `Swap` (två parametrar, de tal som skall byta plats), `Sort` (en parameter, fältet med talen) och `Put` (en parameter, fältet med talen). Vilken typ av underprogram det skall vara får du avgöra. Välj det som är **lämpligt**!

Det är skönt att slippa hitta på och skriva in en stor datamängd var gång programmet ska testas. För att åstadkomma den ursprungliga datamängden i fältet kan det därför vara lämpligt att använda en slumptalsfunktion. Om du inte har information om hur detta görs frågar du assistenten.

Del B:

Du skall nu skriva ett nytt program som genererar 40 slumptal och lagrar dem i ett fält som **representerar** två dimensioner (den ena dimensionen 20 och den andra 2). Detta fält representerar i programmet 20 data där varje data är två heltal (ett talpar).

Lämpligt är att kopiera del A och sen modifiera den kod du redan gjort. Det kommer inte att bli (skall inte bli) speciellt stora modifieringar. OBS! Du skall INTE modifiera själva sorteringsproceduren! Behövs det något så lägger du till fler underprogram som stödjer det som saknas (dock inte inne i sorteringproceduren.

Uppgiften är att först skriva ut dessa data (talparen) sedan sortera dem i stigande ordning. Talparen skall vara sorterade i första hand på det andra talet och i andra hand på det första talet. Till sist skall programmet skriva ut de sorterade talparen på skärmen (ett talpar per rad).

Här följer ett exempel på utskrift av 4 sorterade talpar (i uppgiften skall det bli 20 talpar):

```
5 1
4 2
2 3
4 3
```

Del C:

Du skall nu lägga till möjligheten att styra hur många data som skall lagras i ditt fält. Dels skall man kunna ändra antalet data och dels vilken typ av data som skall lagras (dock vet vi att det alltid är ett antal heltal i ett data så egentligen är det mer dimensionerna vi är ute efter att modifiera).

För att styra antalet data skall du ta in argument (parametrar) från kommandoskalet (terminalen) där du startar programmet. När du startar programmet anger du antal rader respektive kolumner på kommandoraden efter programnamnet. I terminalen kan det se ut på följande sätt om ditt program heter Sort_Data_C och ligger i filen sort_data_c.adb som sedan kompilerats til sort_data_c:

```
sort_data_c   5   3
```

Detta anger att det skall vara 5 rader med trippletter (3 heltal per rad som hör ihop).

Utskriften skulle då kunna bli:

```
1 10 1
3 8 2
2 10 2
10 5 5
1 3 10
```

För att lösa detta måste man tänka till var man lägger sina typdefinitioner. Det går inte att lägga dem i huvudprogrammet längre.

TIPS: Kopiera del B till ny fill så att du kan återgå till en sparad version om det går fel.

Det paket som kan vara bra att ha tillgång till heter Ada.Command_Line. I detta finns det två rutiner som är riktigt bra att ha.

```
function Argument_Count return Natural;
function Argument(Number : Positive) return String;
function Command_Name return String;
```

Dessa tre ger som resultat hur många argument som skickats till programmet, argumentet som skickats som nummer Number i argumentlistan samt vad programmet som startades heter.

I ditt program skall du kontrollera att de argument som skickas till programmet är rimliga och om de inte är detta skall du skriva ut ett felmeddeande och avsluta programmet utan att utföra det som borde gjorts om det var korrekta indata. Två exempel på detta:

```
> sort_data_c   0   3
Det måste vara minst 2 rader för att kunna sortera data.

> sort_data_c   20  -2
Det kan inte vara mindre än 1 data per rad i datamängden.
```

Del D: (frivillig)

Det sista programmet som skall skivas i denna laboration är en variant av ”high score”-lista med 20 resultat och deras rank. Du skall se till att 20 resultat slumpas fram och samtidigt nollställa den rank som senare skall räknas ut. Därefter skall rank för respektive resultat beräknas. Efter detta skall resultaten (och rank) skrivas ut på skärmen osorterade och därefter sorteras (enligt nedan). Till sist skall resultaten (och rank) skrivas ut på skärmen igen. Utskriften skall se ut enligt exemplen nedan.

Även här är det lämpligt att kopiera det som gjorts sen tidigare och modifiera. Det skall inte bli stora ändringar här heller gentemot t.ex. del B.

De som tävlar kan optimera sina resultat på två olika sätt och resultatet för en tävlande blir ett talpar (X, Y). Antingen är man bäst på sätt X eller så är man bäst på sätt Y (det går i vissa fall att vara bäst på båda sätten samtidigt). Optimerar man X kan det hända att Y blir lidande och vice versa.

När man tävlar rankas man dessutom enligt ett speciellt förfarande och detta går till på följande vis. Den rank en tävlande får är endast beroende på hur många som är bättre än honom/henne. Med bättre menas att antingen ’både X och Y är mindre’ eller att antingen ’X är mindre och Y är lika’ eller att ’X är lika och Y är mindre’. Ranken den tävlande får är R=antal tävlande som har bättre resultat + 1. Om man antar att man har två resultat (x_1, y_1) och (x_2, y_2) så kan man uttrycka att (x_1, y_1) är bättre än (x_2, y_2) om $x_1 \leq x_2$ och $y_1 < y_2$ eller $x_1 < x_2$ och $y_1 \leq y_2$.

Sorteringen skall ske på så sätt att man i första hand sorterar på rank och i andra hand på X-resultatet.

Exempel på utskrift (med endast 4 resultat för att visa principen):

```
Rank    Resultat (före sortering)
3       8 3
1       8 2
2       9 2
1       7 3
Rank    Resultat (efter sortering)
1       7 3
1       8 2
2       9 2
3       8 3
```

Ovan ser vi att (7, 3) och (8, 2) inte har några resultat som är bättre => rank 1, talparet (9, 2) är sämre än (8, 2) => rank 2 och till sist har vi (8, 3) som är sämre än både (7, 3) och (8, 2) => rank 3.

Ett annat exempel på utskrift (fundera på varför det inte finns med någon med rank 2):

```
Rank    Resultat (före sortering)
3       3 3
1       3 2
1       2 3
Rank    Resultat (efter sortering)
1       2 3
1       3 2
3       3 3
```

För att lagra informationen i programmet denna gång behövs ett fält med 20 x 3 tal (taltripplar kan vi kalla dem). Varje taltrippel består av resultaten (två tal) och en rank (ett tal).