2017-10-31.kl.17-21

Tentaupplägg

TIPS 1:

Läs igenom ALLA uppgifterna. Välj den du känner är lättast först. Det kan gärna ta 10-20 minuter. Försök skriva saker som kan vara problem i uppgifterna. Är det något du absolut kommer att fastna på så kanske det är fel uppgift att ge sig på. Tiden du lägger på att noga läsa uppgifterna tjänar du in på att välja rätt uppgift.

TIPS 2:

Kolla ibland till kommunikationsfönstret. Det kan ha kommit information till alla utan att ni skickat in en fråga. Kanske gäller det dig också (d.v.s. den uppgift du jobbar med).

TIPS 3:

Om ni har problem med kompilator, Emacs eller annat som INTE har med uppgifterna att göra, räck upp handen så kommer en assistent. Detsamma gäller om hur man kopierar givna filer
" cp given_files/* . " eller liknande.

Frågor om själva uppgifterna tar vi i första hand via tentasystemet.

I körexemplen har vi markerat det som användaren matar in på tangentbordet med ***fet och kursiverad*** stil. Tänk på att körexemplen bara är *ett* exempel på när programmet körs. Testa ditt program noga och tänka över hur programmet skall fungera vid andra indata.

Vi hinner normalt sett inte svara på frågor de sista 10 minuterna på tentan. Då ägnar vi all tid åt att rätta uppgifter för att alla skall hinna få svar innan ni går hem.

Betygsgränser:	**Tid**	**TekFak**	**FilFak**
1 poäng	21:00	Betyg 3	Betyg G
2 poäng	20:30	Betyg 4	
2 poäng	19:30		Betyg VG
2 poäng	19:00	Betyg 5	
>=3 poäng	20:30	Betyg 5	
>=3 poäng	16:30		Betyg VG

OBS: Delpoäng delas inte ut på uppgifterna. För att få poäng på en uppgift måste man alltså lösa uppgiften helt (och enligt specifikation).

Lycka till med tenterandet och hoppas att alla får G på minst en uppgift idag.
M.v.h.
/Torbjörn (examinator)

2017-10-31.kl.17-21

Uppgift 1 - Spelmarker [1p]

När man spelar kort så kan man ha spelmarker (brukar kallas chips) av olika färger (som i bilden). Olika färger indikerar olika värde. Du skall skriva ett program där användaren får ange hur mycket chips den har totalt och sedan skall programmet mata ut vilken uppsättning detta motsvarar om man alltid "växlar" så att man får så högvärderade färger som möjligt. Användaren matar även in vad de olika färgerna är värda (matas alltid in från störst till minst). Vi utgår ifrån att det är sex färger, men du skall göra din lösning generellt så det inte är svårt att ändra i koden och utöka antalet färger. Vi utgår ifrån att användaren alltid matar in '1' som sista värde.

Körexempel 1:

```
Mata in totalt antal chips: 1234
Mata in värden på färger (från störst till minst): 200 100 50 10 5 1

Du får:
6st 200-marker
3st 10-marker
4st 1-marker
```

Körexempel 3:

```
Mata in totalt antal chips: 100
Mata in värden på färger (från störst till minst): 6 5 4 3 2 1

Du får:
16st 6-marker
1st 4-marker
```

Körexempel 3:

```
Mata in totalt antal chips: 666
Mata in värden på färger (från störst till minst): 37 22 13 5 3 1

Du får:
18st 37-marker
```

Uppgift 2 - Spelkort [1p]

Det finns ett givet huvudprogram game.adb som behöver kompletteras med en datatyp för att hantera ett spelkort. Du skall i denna uppgift lägga till den kod som behövs för att definiera datatypen Card_Type som representerar ett spelkort. Ett spelkort har två relevanta delar:

- En valör (siffrorna 2-10, Knekt, Dam, Kung, Äss)
- En färg (Spader, Hjärter, Ruter, Klöver)

Du skall utöver detta lägga till fyra underprogram som hanterar datatypen, en Get, en Put, en Swap och en operator "<". Inläsnings- och utmatningsformat för korten skall vara:

[valör][färg]

Där valören är ett heltal som alltid skrivs ut med två tecken. Vi använder 11 för Knekt, 12 för Dam, 13 för Kung och 14 för Äss. Och färgen är tecknen 'S', 'H', 'D', 'C' (*spades*, *hearts*, *diamonds* och *clubs)*. Ett par exempel:

```
13C  Klöver Kung
 2H  Hjärter två
14D  Ruter Äss
10S  Spader tio
11H  Hjärter Knekt
```

För operatorn "<" skall man betrakta kortens valörer. Om (och endast om) ett kort A har lägre valör än ett annat kort Bs valör, är A mindre än B. Annars är A *inte* mindre än B.

Swap skall byta innehållet på de två parametrar som skickas till underprogrammet.

Körexempel 1:

```
Mata in ett kort: 13C
Mata in ett till kort: 14D
13C är mindre än 14D.
```

Körexempel 2:

```
Mata in ett kort: 6S
Mata in ett till kort: 3H
 3H är mindre än  6S.
```

OBS: Datatypen och underprogrammen i uppgift 4 är helt annorlunda än den datatyp och de underprogram som du skall skapa i denna uppgift.

Uppgift 3 - Rekursion [2p]

Skriv ett program som låter användaren mata in ett heltal och sedan matar ut en figur vars storlek beror på heltalet. Programmet skall fungera för godtyckligt stora heltal. Din lösnings skall vara **rekursiv**, inga loopar tillåtna.

Körexempel 1:

```
Mata in storlek: 1
\-\
/-/
```

Körexempel 2:

```
Mata in storlek: 2
\-\-\
 \ \ \
 / / /
/-/-/
```

Körexempel 3:

```
Mata in storlek: 3
\-\-\-\
 \ \ \ \
  \ \ \ \
  / / / /
 / / / /
/-/-/-/
```

Körexempel 4:

```
Mata in storlek: 4
\-\-\-\-\
 \ \ \ \ \
  \ \ \ \ \
   \ \ \ \ \
   / / / / /
  / / / / /
 / / / / /
/-/-/-/-/
```

Körexempel 5:

```
Mata in storlek: 6
\-\-\-\-\-\-\
 \ \ \ \ \ \ \
  \ \ \ \ \ \ \
   \ \ \ \ \ \ \
    \ \ \ \ \ \ \
     \ \ \ \ \ \ \
     / / / / / / /
    / / / / / / /
   / / / / / / /
  / / / / / / /
 / / / / / / /
/-/-/-/-/-/-/
```

TIPS: Skriv en lösning med loopar först, omvandla sedan gradvis till rekursiva underprogram.

2017-10-31.kl.17-21

Uppgift 4 - Poker [2p]

På filen *poker_for_straight.adb* finns ett huvudprogram som använder en datatyp, *Card_Set_Type*, för att hantera ett pokerspel. I poker använder man en vanlig kortlek och försöker få den bästa uppsättningen fem kort. Korten kan kombineras ihop i olika kombinationer, t.ex. par, tvåpar, triss, stege, färg o.s.v. Den givna datatypen bygger på att man har en tabell där man "kryssar" i vilka kort som ingår i uppsättningen (se bilden nedan).

	A	2	3	4	5	6	7	8	9	10	J	Q	K	Sum
♠					1			1						2
♥									1				1	2
♦														0
♣		1								1			1	3
Sum	0	1	0	0	1	0	0	1	1	1	0	0	2	

De blanka rutorna har värde 0.

Denna uppsättning har ett par i kungar.

Det som är bra med denna datatyp är att man kan representera uppsättningar som har fler kort än bara fem. Detta är intressant för vissa varianter av poker, t.ex. Texas Hold'Em, där man tar den bästa 5-kombinationen (kallas för *handen*) av 7 kort. Sum-kolumnen och sum-raden är extra rader och kolumner som kan användas för att summera hur många kort av en viss färg, eller valör som finns i uppsättningen.

I denna uppgift skall du lägga till en funktion (och ett anrop till denna) som tar en variabel av datatypen Card_Set_Type och avgör om uppsättningen innehåller en *stege*. En stege är fem kort i följd, t.ex. 4 5 6 7 8. Korten behöver inte ha samma färg. Uppsättningen kommer bestå av sju kort. Detta innebär att det t.ex. kan finnas en längre följd än 5, men det räknas ändå som stege. Tänk på att äss ('A' i tabellen) både kan räknas som 1 och 14 i poker. Din funktion skall returnera sant om uppsättningen har en stege, och falskt annars. Här kommer ett par exempel:

♠A ♥A ♠K ♥K ♦2 ♣2 ♠7 => Stege finns ej.

♠A ♥A ♠K ♥Q ♦J ♣2 ♠7 => Stege finns ej (följd på 4 kort räknas inte).

♠A ♥A ♠K ♥Q ♦J ♣10 ♠7 => Stege finns! (10, J (11), Q(12), K(13), A(14))

♠A ♥4 ♠K ♥3 ♦2 ♣10 ♠5 => Stege finns! (A(1), 2, 3, 4, 5)

♠6 ♥4 ♠7 ♥3 ♦2 ♣10 ♠5 => Stege finns! (2, 3, 4, 5, 6 eller 3, 4, 5, 6, 7)

♠6 ♥4 ♠7 ♥3 ♦2 ♣8 ♠5 => Stege finns! (2, 3, 4, 5, 6 eller 3, 4, 5, 6, 7 eller 4, 5, 6, 7, 8)