2017-10-31.kl.13-17

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	17:00	Betyg 3	Betyg G
2 poäng	16:30	Betyg 4	
2 poäng	15:30		Betyg VG
2 poäng	15:00	Betyg 5	
>=3 poäng	16:30	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.13-17

Uppgift 1 - Tydlighet [1p]

Ibland när man skriver saker så måste man vara tydlig. För att folk verkligen skall förstå krävs det att man är extra TYDLIG. Ibland behöver man verkligen vara extremt

TTTTTTYYYYYYDDDDDLLLLLLLLIIIIIIIIIIIIGGGGG!!!!!!!!!!!!!!!!!!!

Skriv ett program som läser in ett ord på sex tecken och sedan gör detta ord "tydligt". Detta görs genom att läsa in 7 icke-negativa heltal. Varje heltal indikerar hur många gånger varje bokstav i ordet skall skrivas ut. Det sista heltalet anger antalet utropstecken.

OBS: Lös problemet generellt, det skall vara lätt att ändra i koden så att programmet klarar av ord som är t.ex. 7 tecken (eller 100).

Körexempel 1:

```
Mata in ett ord (6 tecken): TYDLIG
Mata in heltal (7 stycken): 3 5 4 2 8 5 3

TTTYYYYYDDDDLLIIIIIIIIGGGGG!!!
```

Körexempel 2:

```
Mata in ett ord (6 tecken): CAROLA
Mata in heltal (7 stycken): 1 2 3 4 5 6 7

CAARRROOOOLLLLLAAAAAA!!!!!!!
```

Körexempel 3:

```
Mata in ett ord (6 tecken): JOANNA
Mata in heltal (7 stycken): 1 0 0 0 0 10 5

JAAAAAAAAAA!!!!!
```

2017-10-31.kl.13-17

Uppgift 2 - Spelmarker [1p]

Det finns ett givet huvudprogram chips.adb som behöver kompletteras med en datatyp för att hantera ett gäng spelmarker. Du skall i denna uppgift lägga till den kod som behövs för att definiera datatypen Chips_Type. En sådan har två relevanta delar:

- Ett antal (ett heltal)
- En färg (*white*, *red*, *blue*, *green*, *black*, eller *gold*)

Färgerna representerar olika värden. Vita marker är värda 1, röda 5, blå 10, gröna 20, svarta 100 och guldfärgade 200.

Du skall utöver detta lägga till tre underprogram som hanterar datatypen, en Get, en Put, och en operator "+".

Inläsnings- och utskriftsformat för spelmarkerna skall vara:

[färg] [antal]

Färgen kan endast vara de som anges ovan och matas alltid in med exakt fem tecken (man fyller ut med blanksteg för t.ex. red). I den givna koden finns en funktion som omvandlar en sådan sträng till motsvarande värde. Det finns också en funktion som omvandlar ett värde till motsvarande färg. Du bör använda dessa i din lösning.

Operatorn + skall summera två parametrar av typen Chips_Type och returnera summan som en ny Chips_Type. Operatorn skall följa denna algoritm:

1. Hitta den parameter med färg som är minst värd. Kalla den parametern för A, den andra för B.
2. Sätt färgen i summan till As färg.
3. Sätt antalet i summan till As antal + (Bs antal * (värdet på Bs färg / värdet på As färg)).
4. Returnera summan.

Körexempel 1:

```
Input some chips: blue  13
Input some more chips: red   4

The sum is red   30.
```

*(Förklaring: 13 blå spelmarker är värda 13 * 10 = 130. Detta motsvarar 130 / 5 = 26 röda marker eftersom röda marker är värda 5. Tillsammans med de fyra röda blir det 30 röda marker totalt.)*

Körexempel 2:

```
Input some chips: gold  1
Input some more chips: white 64

The sum is white 264.
```

Körexempel 3:

```
Input some chips: green  4
Input some more chips: green 16

The sum is green 20.
```

2017-10-31.kl.13-17

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.13-17

Uppgift 4 - Poker [2p]

På filen *poker_for_full_house.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 *kåk*. En kåk består av en triss (tre kort av samma valör) och ett par (två kort av samma valör). **OBS:** paret får inte ingå i trissen. Uppsättningen kommer bestå av sju kort. Detta innebär att det t.ex. kan finnas två trissar, men eftersom *handen* bara få bestå av 5 kort så kommer en av dem bara räknas som ett par. Ditt underprogram skall returnera *sant* om uppsättningen har en kåk, och falskt annars. Här kommer lite exempel på uppsättningar:

♠A ♥A ♠K ♥K ♦2 ♣2 ♠7 => Kåk finns ej. (Ingen triss).

♠A ♥A ♣A ♥K ♦2 ♣6 ♠7 => Kåk finns ej. (Inget par (om inte ingår i en triss)).

♠A ♥A ♣A ♥K ♦2 ♣2 ♠7 => Kåk finns!

♠A ♥A ♣A ♥K ♦K ♣2 ♠2 => Kåk finns! (Triss i äss och par i kungar, eller triss i äss och par i tvåor).

♠A ♥A ♣A ♥2 ♦2 ♣2 ♠7 => Kåk finns! (triss i äss och par i tvåor, eller par i äss och triss i tvåor).

♠A ♥A ♣A ♦A ♦2 ♣2 ♠7 => Kåk finns! (egentligen fyrtal, som är ännu bättre, men det finns ändock en kåk där (tre äss och paret)).