2018-11-06 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	20:30		Betyg VG

Bonustid från laborationer tillkommer till dessa tidsgränser. Tiden för betyg 3/G överstiger dock inte fyra timmar.

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

2018-11-06 kl. 17-21

Uppgift 1 - Bordsplacering [1p]

Skriv ett program som ritar ut en borsplacering för ett antal personer. Användaren matar in ett godtyckligt positivt heltal N och programmet skriver ut bordsplaceringen, med numrerade platser, enligt nedanstående körexempel. Om det är ett udda antal personer så blir en plats ledig (i nedre vänstra hörnet).

Körexempel 1:

```
Mata in antal personer: 4

+-------+
|  1  2 |
|  4  3 |
+-------+
```

Körexempel 2:

```
Mata in antal personer: 8

+-------------+
|  1  2  3  4 |
|  8  7  6  5 |
+-------------+
```

Körexempel 3:

```
Mata in antal personer: 5

+----------+
|  1  2  3 |
|     5  4 |
+----------+
```

Körexempel 4:

```
Mata in antal personer: 19

+------------------------------+
|  1  2  3  4  5  6  7  8  9 10 |
|    19 18 17 16 15 14 13 12 11 |
+------------------------------+
```

TIPS: Se upp för fullständig uppräkning, lös generellt...

2018-11-06 kl. 17-21

Uppgift 2 - Flyg [1p]

På den givna filen test_flights.adb finns ett huvudprogram som vill använda sig av datatypen *Flight_Type*. En Flight_Type är en datatyp som innehåller dels ett flygbolag (tre tecken, tex. "SAS" eller "KLM") och ett nummer (ett heltal i intervallet [1, 9999]). Du skall i denna uppgift skapa den datatyp och de underprogram som behövs för att kompilera och köra programmet. De underprogram som behövs är:

Ett underprogram **Get**, som tar en parameter av typen Flight_Type och läser in data från tangentbordet till den. Formatet är alltid BLG NUMMER där BLG är flygbolaget och NUMMER är heltalet. Vi antar att flyget matas in enligt korrekt format. Ingen felhantering krävs.

Ett underprogram **Put**, som tar en parameter av typen Flight_Type och skriver ut den på skärmen. Formatet är samma som inmatningsformatet.

Ett underprogram **Next_Flight**, som tar en parametrar av typen Flight_Type och returnerar *nästa flyg*. Nästa flyg fås genom att addera 1 till flygets nummer (och det är fortfarande samma bolag). Om flyget som kommer in som parameter har nummer 9999 så skall undantaget No_More_Flights_Error kastas eftersom inget bolag har ett flyg nr 10000 (eller högre).

Körexempel 1:

```
Mata in flightnummer: KLM 1

Nästa flyg är KLM 2.
```

Körexempel 2:

```
Mata in flightnummer: SAS 945

Nästa flyg är SAS 946.
```

Körexempel 3:

```
Mata in flightnummer: RYA 9999

Det fanns inget nästa flyg!
```

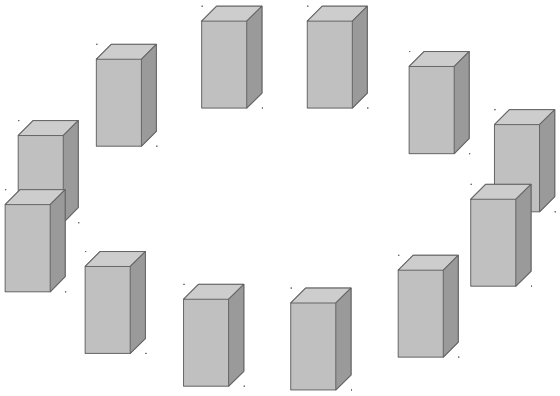
TIPS: Det är OK att göra ett paket för att hantera Flight_Type, men det är inte ett krav.

KRAV: Du skall **inte** modifiera det givna huvudprogrammet mellan begin och end.

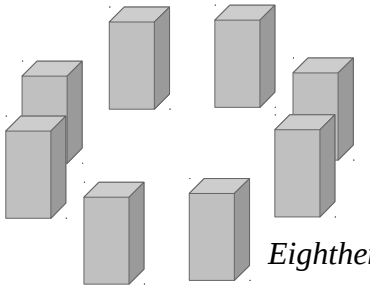
2018-11-06 kl. 17-21

Uppgift 3 - Rüktal's Henges Inc. [2p]

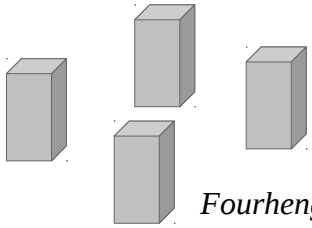
En föregångare till Stonehenge var *Dozenhenge*, som hade exakt 12 stenar i en ring:



Inte lika snyggt kanske, men en förbättring över *Eighthenge*, och klart överlägsen det gamla *Fourhenge*:



Eighthenge



Fourhenge

Druiden Rüktal (som är i henge-branschen) har kommit på att man kan göra riktigt snygga henges om man anordnar stenarna i koncentriska circlar, där varje inre cirkel har fyra färre stenar än den som ligger utanför.. Hövdingen i hans klan har nu skrytit vitt och brett över den nya pampiga hengen och därmed har Rüktal många andra hövdingar som vill ha lika dana henges i sina trakter. Hövdingarna har olika mycket sten och olika stora ytor som de kan bygga sina henges på. Rüktal behöver därför ett datorprogram som beräknar vilka henges som kan byggas.

Användaren matar in det maximala antalet tillgängliga stenar och hur stor den yttersta hengen kan vara. Programmet skall sedan ange vilka henges som skall byggas. Rüktals henges har alltid en multipel av fyra stenar i varje ring, så att det blir riktigt snyggt.

Körexempel 1:

```
Mata in totala antalet stenar: 21
Mata in maximal omkrets för yttersta henge: 14
Bygg en 12-henge utanför en 8-henge.
1 sten(ar) över.
```

Körexempel 2:

```
Mata in totala antalet stenar: 253
Mata in maximal omkrets för yttersta henge: 71
Bygg en 68-henge utanför en 64-henge utanför en 60-henge utanför en
56-henge.
5 sten(ar) över.
```

KRAV: Ditt program skall vara rekursivt. Inga loopar får användas.

2018-11-06 kl. 17-21

Uppgift 4 - Spaltade argument [2p]

Skriv ett program ***my_columns*** som tar in ett godtyckligt antal argument från kommandoraden. Argumenten skall sedan skrivas ut i en högerjusterad kolumn. Bredden på kolumnen bestäms av längsta ordet.

Du får utgå ifrån att användaren alltid matar in kommandoradsargumenten. Ingen kontroll behövs.

Körexempel 1 (från kommandoraden, '$' är prompten):

```
$ my_columns apple banana orange grape
 apple
banana
orange
 grape
```

Körexempel 2 (från kommandoraden, '$' är prompten):

```
$ my_columns apple banana orange grape taco
 apple
banana
orange
 grape
  taco
```

Körexempel 3 (från kommandoraden, '$' är prompten):

```
$ my_columns AX BY CZ MAN CAR VOICE
   AX
   BY
   CZ
  MAN
  CAR
VOICE
```

TIPS: Funktionerna *Argument(N)* och *Argument_Count* finns i paketet Ada.Command_Line. Funktionen *Argument* returnerar det N:te argumentet som programmet startades med (som en sträng). Funktionen *Argument_Count* returnerar antalet argument som programmet startades med.