

# 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.

<b>Betygsgränser:</b>	<b>Tid</b>	<b>TekFak</b>	<b>FilFak</b>
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

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)

## Uppgift 1 - Bordsplacering [1p]

Skriv ett program som ritar ut en bordsplacering 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 övre högra hörnet).

### Körexempel 1:

Mata in antal personer: **4**

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

### Körexempel 2:

Mata in antal personer: **8**

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

### Körexempel 3:

Mata in antal personer: **5**

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

### Körexempel 4:

Mata in antal personer: **19**

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

**TIPS:** Se upp för fullständig uppräknig, lös generellt...

## Uppgift 2 - Priser [1p]

På den givna filen `test_prices.adb` finns ett huvudprogram som vill använda sig av datatypen `Price_Type`. En `Price_Type` är en datatyp som innehåller dels ett belopp (ett heltal) och en valuta (tre tecken, t.ex. "SEK" eller "USD"). 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 `Price_Type` och läser in data från tangentbordet till den. Formatet är alltid `BELOPP VAL` där `BELOPP` är ett icke-negativt heltal och `VAL` är valutan. Vi antar att priset matas in enligt korrekt format. Ingen felhantering krävs.

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

Ett underprogram `<`, som tar två parametrar av typen `Price_Type` och jämför om den ena är mindre än den andra. Priser jämförs endast på belopp. Om priserna har olika valutor skall undantaget `Uncomparable_Prices_Error` kastas.

### Körexempel 1:

```
Mata in första pris: 300 SEK
Mata in andra pris: 500 SEK
```

300 SEK var det minsta av dessa två pris.

### Körexempel 2:

```
Mata in första pris: 20 USD
Mata in andra pris: 1000 EUR
```

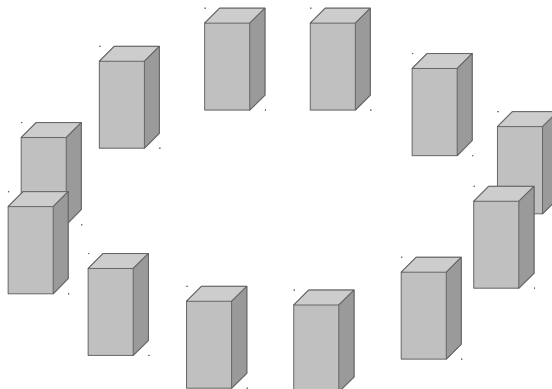
Det gick inte att jämföra dessa två priser!

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

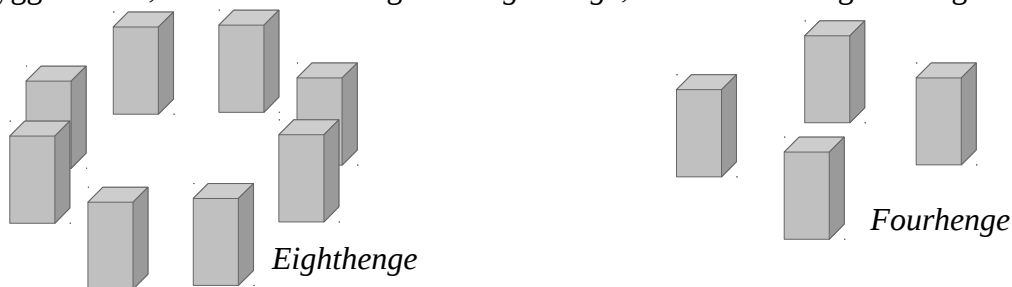
**KRAV:** Du skall **inte** modifiera det givna huvudprogrammet mellan `begin` och `end`.

## Uppgift 3 - Rüktaal'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*:



Druiden Rüktaal (som är i henge-branschen) har kommit på att man kan göra riktigt snygga henges om man anordnar stenarna i koncentriska cirklar, 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 Rüktaal 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. Rüktaal 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. Rüktaals henges har alltid en multipel av fyra stenar i varje ring, så att det blir riktigt snyggt. Observera att det ibland blir stenar över.

### Körexempel 1:

Mata in totala antalet stenar: **21**

Mata in maximal omkrets för yttersta henge: **14**

Bygg en 8-henge i en 12-henge.

### Körexempel 2:

Mata in totala antalet stenar: **253**

Mata in maximal omkrets för yttersta henge: **71**

Bygg en 56-henge i en 60-henge i en 64-henge i en 68-henge.

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

## Uppgift 4 - Omvända argument [2p]

Skriv ett program *my\_reverse* som tar in ett godtyckligt antal argument från kommandoraden. Programmet skall sedan skriva ut dessa argument i omvänd ordning och varje argument skall vara spegelvänt (så att första tecknet kommer sist, andra tecknet kommer näst sist, o.s.v.).

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

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

```
$ my_reverse apple banana orange grape  
eparg  
egnaro  
ananab  
elppa
```

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

```
$ my_reverse car racecar palindrome  
emordnilap  
racecar  
rac
```

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

```
$ my_reverse Area51  
15aerA
```

**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.