

## UPPGIFT 1 – MULTIPLIKATION

$$\begin{array}{r} 715 \\ 46 \\ \hline 32890 \end{array} \qquad \begin{array}{r} *1* \\ ** \\ \hline ***** \end{array}$$

FIGURE 1.

Till vänster i figuren ser Du en multiplikation där den ena faktorn innehåller tre siffror, den andra två och produkten fem. Dessutom ser Du att varje siffra 0 till 9 används precis en gång.

Till höger i figuren ser du en liknande uppställning, där alla utom en siffra har ersatts med asterisker. Om asteriskerna ska ersättas av siffror, så att varje siffra från 0 till 9 finns med precis en gång och så att faktorerna innehåller tre respektive två siffror och produkten fem, så finns det bara en lösning – den till vänster.

Skriv ett program som tar reda på antalet uppställningar, liknande den till höger, där *en* siffra är given någonstans och där det finns exakt en lösning. Antalet siffror i de tre talen ska, precis som i exemplet, hela tiden vara tre, två och fem. Inget av de tre talen får inledas med siffran 0.

**Indata:** -

**Utdata:** Ett tal som anger antalet uppställningar med unika lösningar

## UPPGIFT 2 – KULOR I KVADRAT

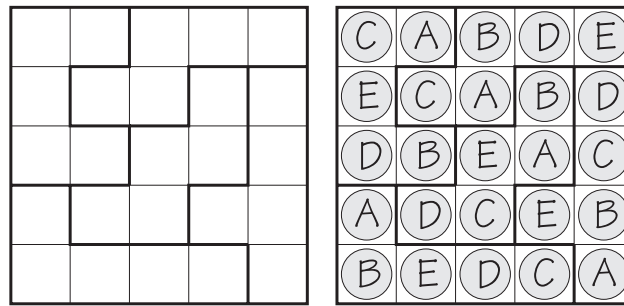


FIGURE 2.

I figuren till vänster ser Du en kvadrat. Kvadraten är indelad i fem områden. Varje område består av 5 mindre kvadrater.

Till uppgiften hör 25 kulor i 5 färger, 5 av varje färg, Azurblå (A), Beige (B), Citrongul (C), Djupröd (D), Ebenholtssvart (E).

Skriv ett program som utifrån en given kvadrat placerar ut de 25 kulorna i kvadraten så att

- det finns en kula av varje färg i varje rad.
- det finns en kula av varje färg i varje kolumn.
- det finns en kula av varje färg i vart och ett av kvadratens fem områden.

Till höger i figuren ser du en lösning för just denna kvadrat.

**Indata:** Programmet ska fråga efter namnet på en fil som innehåller information om kvadratens utseende. Filen består av fem rader med fem tal (1 - 5) på varje rad. Filen till exemplet har följande utseende:

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

**Utdata:** Fem rader med fem bokstäver i varje rad. Vårt exempel ger:

```
C A B D E
E C A B D
D B E A C
A D C E B
B E D C A
```

Programmet behöver bara skriva ut en av många lösningar.

## UPPGIFT 3 – ENARMAD BANDIT

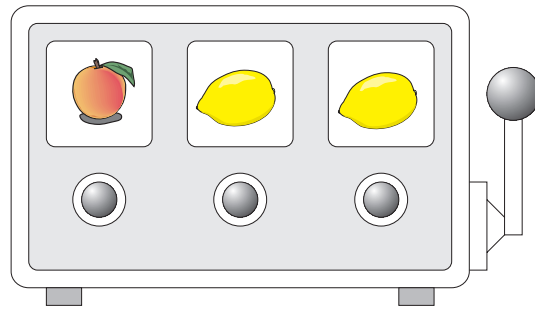


FIGURE 3.

Figuren visar en *Enarmad Bandit* med lite speciellt utseende. De tre hjulen kan bara visa två olika symboler *persika* (P) och *citron* (C). När spelaren drar i armen till höger snurrar ett, två eller tre hjul, beroende inställning. Varje gång automatens hjul visar tre identiska symboler får spelaren en *vinst*.

Innan spelaren drar i spaken för ett nytt spel har han möjlighet att hålla kvar ett eller två hjul, genom att trycka på knappen under motsvarande fönster. Övriga hjul sätts igång och stannar så småningom på en symbol. Detta förfarande kan upprepas inför varje spel och på det sättet kan ett hjul stanna på samma position under en flera spel.

Vid detta speciella tillfälle har spelaren fått reda på maskintillverkarens hemlighet. Han vet redan nu vilka symboler de olika hjulen i tur och ordning kommer att visa upp efter var och en av  $n$ , ( $n \leq 25$ ) *snurrningar*.

Skriv ett program som bestämmer den maximala vinsten för spelaren efter  $n$  spel.

**Indata:** Programmet ska ta emot namnet på en fil som innehåller data om vad de tre hjulen kommer att visa. Filen inleds med ett tal som anger *antalet spel* som kommer att göras. Därefter följer tre rader, en för varje hjul, som anger vad hjulen kommer att visa. Ett exempel

```
11
PCCPPPCCPCCP
PCPCPCPPPPC
CPCPPCPCPPC
```

Första tecknen i varje rad anger vad hjulen visar *från start*. Detta är aldrig en vinstkombination och räknas heller inte som ett spel. Filen innehåller alltid minst så många symboler, för varje hjul, som kan komma att behövas.

Stoppa till exempel spelaren hjul 1 och 2 i första spelet kommer alla tre hjulen att visa persikor. Låter han sedan alla hjulen snurra i nästa spel får han vinst igen genom tre citroner ...

**Utdata:** Exemplet ovan skulle ge följande utskrift:

```
Spelaren kan som mest vinna i 9 av de 11 spelen.
```

## UPPGIFT 4 – TORNET

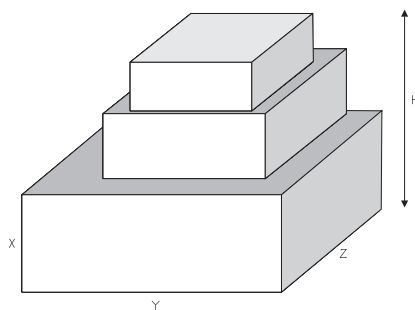


FIGURE 4.

Nu ska vi bygga ett torn, så högt som möjligt! Till vårt förfogande har vi ett antal  $n \leq 20$  klotsar i form av rätblock. Klotsarnas tre dimensioner är givna i form av taltripplar  $(z, y, z)$  där  $0 < x, y, z \leq 50$ .

För att få placera en klots ovanpå en annan måste basens båda dimensioner hos den övre klotsen vara *strängt* mindre än motsvarande dimensioner hos den underliggande.

Klotsarna kan vändas och vridas för att på bästa sätt uppfylla sitt ändamål.

Skriv ett program som tar emot uppgifter om måtten hos  $n$  klotsar och som sedan beräknar höjden för det högsta torn som går att åstadkomma.

**Indata:** Programmet ska fråga efter namnet på en fil som innehåller geometrin hos  $n$  klotsar. Filens första rad innehåller ett tal som fastställer  $n$ . Därefter följer lika många rader med tre tal, var och en beskrivande en klots.

Ett exempel:

```
8
1 1 3
2 2 2
5 4 2
3 3 4
3 5 3
5 5 3
7 9 5
7 7 7
```

**Utdata:** Programmet ska skriva ut en mening i stil med denna

Det högsta torn som går att bygga har höjden 22

Tornet kan byggas så här med understa biten först och med (längd, bredd, höjd):  $(7,5,9)$ ,  $(5,3,5)$ ,  $(4,2,5)$ ,  $(1,1,3)$

UPPGIFT 5 – TRIANGELN

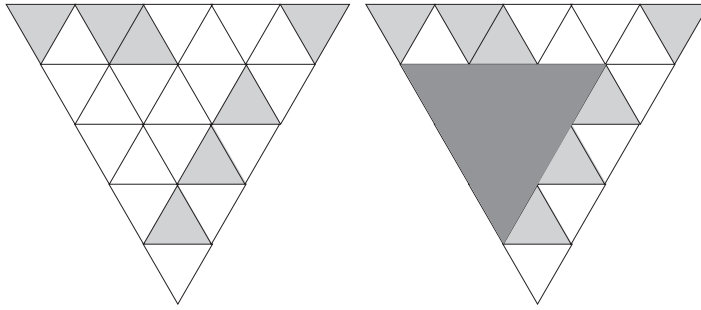


FIGURE 5.

I figuren till vänster ser du en liksidig triangel, som i sin tur består av 25 små trianglar. En del av dessa mindre trianglar är gråa resten är vita.

Du ska skriva ett program som finner arean till den största liksidiga triangel som enbart består av vita småtrianglar och som är en del av den stora triangeln.

Till höger i figuren är denna triangel markerad och det rätta svaret är 9.

**Indata:** Programmet ska ta emot namnet på en fil som innehåller beskrivning av den stora triangeln. Filen inleds med ett tal som anger antalet rader  $n$ , ( $n \leq 30$ ) i filen. Därefter följer  $n$  rader som var och en beskriver de små trianglarna med ett plustecken (+) för grå triangel och minustecken (-) för vit triangel

```
5
+-+-----+
-----+-
---+-
-+-
-
```

Filen inleds alltid med basen – den längsta raden.

**Utdata:** Exempel ovan skulle ge följande utdata:

Den största triangeln består av 9 småtrianglar.

## UPPGIFT 6 – PUSSLET

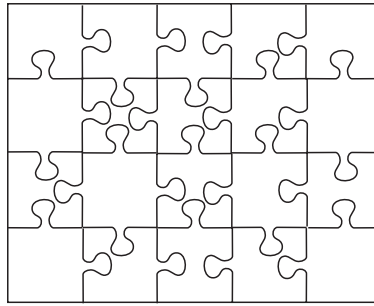


FIGURE 6.

I den här uppgiften ska Du låta datorn lägga pussel. I figuren ovan ser du ett redan färdiglagt pussel innehållande 20 bitar.

Bitarnas utseende finns lagrade i en textfil. Varje rad i textfilen består av fyra tecken, stora bokstäver, enligt tabellen nedan.

R	En rak sida
U	En utbuktning
I	En inbuktning

Beskrivningen av en bit börjar alltid med den *nordliga* sidan, följd av *östlig*, *sydlig* och till sist den *västliga*. Biten i figuren längst upp i hörnet beskrivs följaktligen som RUIR. Ingen bit får vridas eller vändas.

Programmet ska bestämma antalet lösningar. Eftersom alla bitar har olika färger (syns inte här), så får man en ny lösning då två identiska bitar byter plats! Det finns alltid minst en lösning.

**Indata:** Programmet ska ta emot namnet på en fil som innehåller beskrivning av bitarna. Filen inleds med ett tal  $n$ , ( $n \leq 36$ ) som anger antalet bitar följt av lika många rader i godtycklig ordning med fyra tecken per rad. Från exemplet får filen följande utseende:

```
20
RUIR RUUI RIUI RIIU RRIU UURR IIII IIIU UIIU URUU
IIIR UUUU UIII UIUU IRIU UURR IURI UIRI IIRU URRU
```

OBS! Av utrymmesskäl visas här de 20 raderna på endast två!

**Utdata:** Exemplet ovan skulle ge följande utdata:

Det finns 1 lösningar.

## UPPGIFT 7 – KORSORDET

G	L	M	E	G	A	M	E
A	L	A	N	A	L	A	N
G	O	A	D	G	O	L	D
S	E	E	S	S	E	E	S

FIGURE 7.

I detta program ska du rätta till ett skrivfel i ett korsord. Till vänster ser du ett kvadratisk korsord med, som det var tänkt, åtta ord från en engelsk rättstavningsfil.

Det visar sig att två bokstäver i krysset bytt plats. När dessa skiftas tillbaka igen får vi korsordet som vi vill ha det (till höger i figuren).

Skriv ett program som tar emot ett *felaktigt* korsord där programmet också får reda på hur många bokstäver som bytt plats. Programmet ska sedan med hjälp av innehållet i filen ORD.DAT försöka finna det korrekta krysset. Det vill säga ett kryss där alla åtta orden finns i ordfilen.

För att göra det lite enklare så bestämmer vi att ett byte mellan två bokstäver inte får innebära att den nya platsen för en flyttad bokstav ger ett ord som finns i ordfilen.

Antalet bokstäver som kan flyttas är *två*, *tre* eller *fyra*.

**Indata:** Programmet ska fråga efter namnet på en fil som innehåller det felaktiga korsordet. Första raden innehåller ett tal som anger antalet bokstäver som flyttats. Därefter fyra rader med med fyra gemener (små bokstäver) från den engelska alfabetet, a till z, 26 stycken. Filens utseende från ett exempel

```
2
glme
alan
goad
sees
```

**Utdata:** Det korrekta krysset i form av fyra rader med ett fyrbokstavigt ord på varje (som alla finns i ordfilen). Från vårt exempel:

```
game
alan
gold
sees
```

## UPPGIFT 8 – SPRINGARJAKTEN

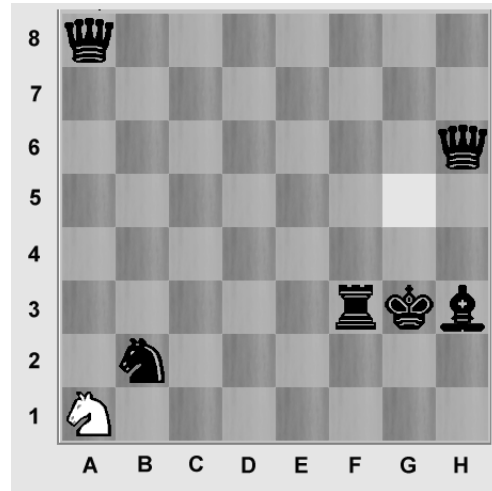


FIGURE 8.

Denna uppgift handlar om ett ovanligt schackproblem. I figuren ser du en vit springare på A1. Springaren ska nu ta sig till den ljusa rutan på G5. Det är bara den vita springaren som flyttas under spelet.

Springaren kan under sin vandring slå svarta pjäser men får aldrig ställa sig på en av svart pjäs hotad ruta. Springaren kan alltså inte flyttas till B3 i första draget i exemplet. Den kan dock stå i slag i utgångsläget.

Skriv ett program som från en given ställning bestämmer det minsta antalet drag den vita springaren behöver göra för att nå sitt mål. Observera att den vita hästen alltid startar på A1.

Det kan finnas flera svarta pjäser av varje typ, även flera kungar. *Dock inga bönder.*

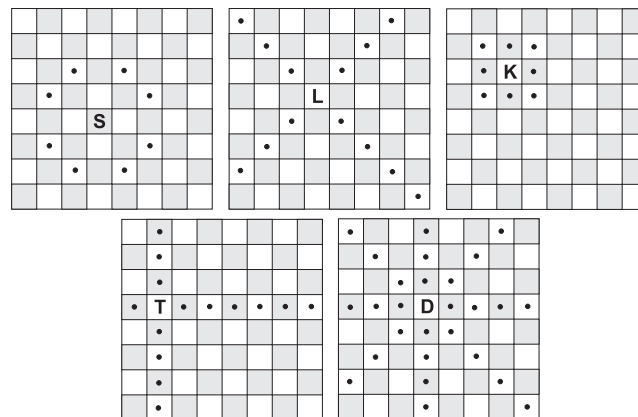


FIGURE 9.

Om Du händelsevis inte känner till hur de olika pjäserna flyttas kan du se det i figuren ovan. Tabellen förklarar förkortningarna

K	Kung
D	Dam
T	Torn
L	Löpare
S	Springare

**Indata:** Programmet ska fråga efter namnet på en fil, som beskriver placeringen av de svarta pjäserna. Ingen svart pjäs kan finnas på A1. Rutan för vandringens mål markeras med asterisk (\*) och tom ruta med punkt(.)

Från exemplet får vi:

```
D.....
.....
.....D
.....*
.....
.....TKL
.S.....
.....
```

**Utdata:**

För den kortaste vandringen till målet behövs 18 drag

Eftersom det är svårt att finna problem som kräver mer än 20 drag bestämmer vi att inget testproblem kommer att kräva fler än 20 drag För att förtydliga problemet ges här också vandringen: C2 D4 B5 C7 A8 C7 D5 E7 G8 H6 G8 E7 C6 B4 A6 C5 E4 G5.