Programmeringsolympiadens final 2009

TÄVLINGSREGLER

- Tävlingen äger rum den 12 mars. Tävlingstiden är sex timmar effektiv tid.
- Tävlingen består av sju uppgifter som samtliga ska lösas genom datorprogram.
- Uppgifterna ska lösas i valfritt programmeringsspråk. Du får byta språk mellan olika uppgifter.
- Tävlingsbidragen lämnas som exekverbara filer i Windows-format (EXE). Dessutom ska källkoden bifogas. För Linux/Mac-användare som använder gcc kan vi göra undantag från regeln med EXE-filer och istället kompilera källkodsfilen själva. Inkludera en kommentar i källkoden med den fullständiga kommandoraden för att kompilera programmet. Detsamma gäller om du använder ett interpreterande språk, t.ex. PHP (ange även version). Om du använder Java, så fungerar class-filen som exekverbar.
- I varje källkodsfil ska finnas en kommentar innehållande namn och skola.
- Lösningarna poängsätts med max 5 poäng per uppgift. Fem tester, med varierande krav hos ditt program, kommer att göras vid rättningen (undantag kan finnas). Möjlighet till delpoäng finns om programmet klarar endast en del av dessa tester. Ingen närmare bedömning av programkoden görs.
- Ingen test av indata behöver göras. Alla testdata följer de specifikationer som givits i uppgiften. Om det trots detta, vid rättningen, uppstår exekveringsfel vid körning av programmet bedöms programmet som felaktigt för det testexemplet.
- Samtliga uppgifter leder fram till program vars exekveringstid bör understiga 5 sekunder på en modern dator. Skulle exekveringstiden för ditt program överskrida denna tid bedöms programmet med 0 poäng för detta testexempel.
- Du har tillgång till de indatafiler som används i uppgiftens exempel.
- Deltagandet är individuellt vilket bland annat innebär att inget utbyte av idéer eller filer får ske under tävlingstiden. Självklart får din dator inte vara kopplad till vare sig internt eller externt nät.
- Hjälpmedel förutom dator med installerade programspråk: Programspråkets manualer, vanlig formelsamling och räknedosa.
- I flera av uppgifterna ska indata läsas från en vanlig ASCII-fil (se nästa sida).
- Tävlingsbidragen ska läggas i roten på utdelad diskett eller i en av läraren angiven hårddiskkatalog. Filerna ska döpas till uppg1...uppg7 med passande filtillägg. Ingen hänsyn tas till andra filer. Var noga med att lämna in den korrekta versionen av ditt program.
- Tips: Det kan vara värt att göra egna indata för att testa ditt program. Även om programmet klarar testexemplet behöver det inte vara korrekt.

LATHUND FÖR INLÄSNING FRÅN FIL

Exempel på hur man i fem språk kan läsa in följande indata från filen `fil.txt`:

```
4 6
3.22 Text
```

Observera att fel kan förekomma.

C

```
#include <stdio.h>
...
int a1, a2;
char word[100];
double d;
FILE *fil=fopen("fil.txt", "rt");
fscanf(fil, "%d %d", &a1, &a2);
fscanf(fil, "%lf %s", &d, word);
```

C++

```
#include <iostream>
using namespace std;
...
int a1, a2;
char word[100];
double d;
ifstream fil("fil.txt");
fil >> a1 >> a2;
fil >> d >> word;
```

Java (J2SE)

```
import java.util.Scanner;
import java.io.File;
...
Scanner sc=null;
sc = new Scanner(new File("fil.txt"));
int a1=sc.nextInt(), a2=sc.nextInt();
double d=sc.nextDouble();
String word=sc.next();
```

Pascal

```
VAR
infile : Text;
a1, a2 : Integer;
d : Double;
word : string[100];
...
Assign(infile, 'fil.txt');
Reset(infile);
Readln(infile, a1, a2);
Readln(infile, d, word);
Close(infile);
```

Basic

```
Dim s,word As String
Dim a1, a2 As Integer
Dim d As Double
Dim sar() As String
...
Open "fil.txt" For Input As #1
Line Input #1, s
sar = Split(s," ")
a1 = val(sar(0))
a2 = val(sar(1))
Line Input #1, s
sar = Split(s," ")
d = val(sar(0))
word = sar(1)
```

Lycka till!

UPPGIFT 1 – TÅRTRESTER

Festen är över. På bordet finns ett antal tallrikar. En del är fulla med tårta, andra är halvfulla eller tomma. De tre värdarna ska till att börja diska efter festen. De ska diska lika många tallrikar var och dessutom vill de ha lika mycket av den överblivna tårtan utan att för den skull flytta tårta mellan tallrikarna.

Om vi till exempel antar att det finns 3 tomma, 6 halvfulla och 9 fulla tallrikar kan de delas upp så här:

	Tomma	Halvfulla	Fulla
Värd 1	0	4	2
Värd 2	1	2	3
Värd 3	2	0	4

Som vi ser får alla 6 tallrikar var att diska. Totalt får de också 4 fulla tallrikar tårta var. Ett annat godkänt sätt vore att alla fick 1 tom, 2 halvfulla och 3 fulla tallrikar.

Skriv ett program som frågar efter antalet tallrikar av varje typ och som presenterar en lösning på hur de ska fördelas. Du kan utgå ifrån att det finns minst en lösning. Om det finns flera lösningar räcker det med att ditt program ger en av dessa. Antalet tallrikar av varje typ kommer att vara mellan 0 och 1000.

Körningsexempel:

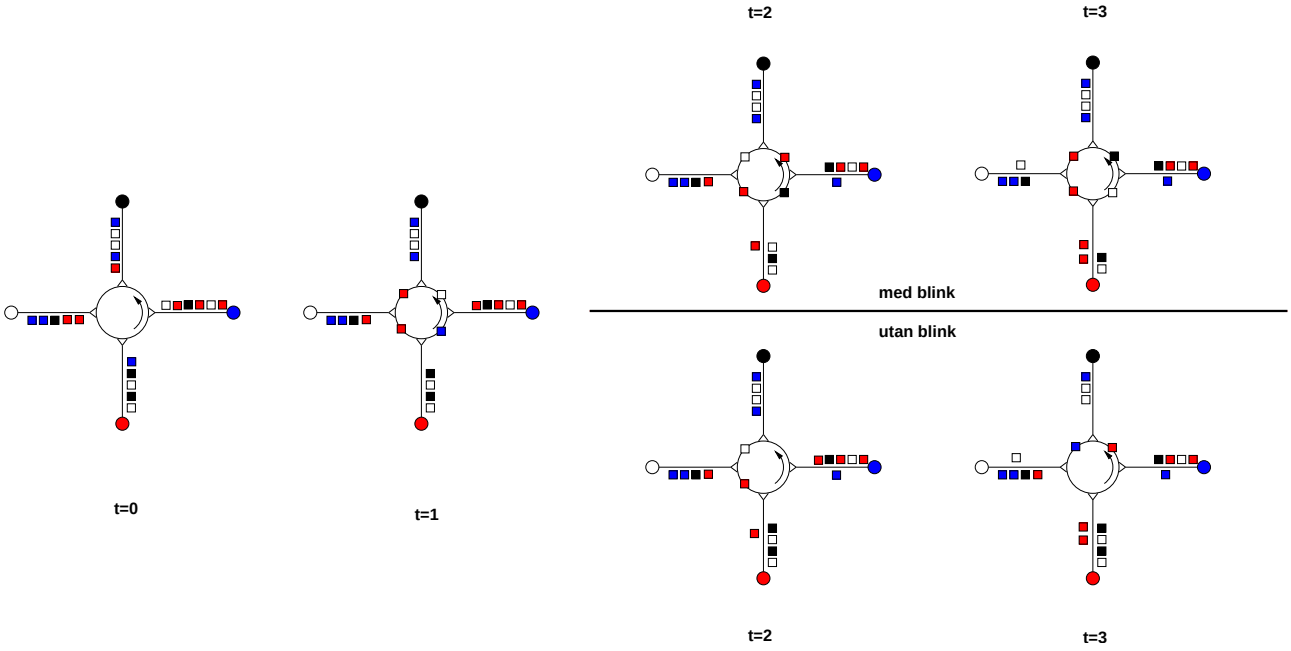
```
Antal tomma ? 1
Antal halvfulla ? 7
Antal fulla ? 7

Värd 1 -> T: 0  H: 3  F: 2
Värd 2 -> T: 0  H: 3  F: 2
Värd 3 -> T: 1  H: 1  F: 3
```

UPPGIFT 2 – RONDELLSIMULERING

Som en del av ett trafikplaneringspaket ska du skriva ett program som simulerar en rondell. Programmet ska beräkna hur lång tid det tar för ett antal bilar att passera rondellen. För att ta hänsyn till bilisters varierande användning av blinkers ska programmet räkna på två extremfall:

- *Med blink*: Alla bilar blinkar när de ska lämna rondellen.
- *Utan blink*: Ingen bil blinkar när den ska lämna rondellen.

Rondellen har N tillfartsvägar som är symmetriskt fördelade runt en cirkel (se figur nedan). Vid tiden 0 finns inga bilar i rondellen men vid varje tillfart finns en kö av bilar. Inga andra bilar tillkommer under simuleringen. Varje bil har en viss tillfartsväg som mål och ska alltså köra moturs (framåt) i rondellen för att komma dit. Mellan varje par av intilliggande tillfartsvägar finns en "position", som används för att underlätta simuleringen (se figuren nedan). Högst en bil kan befinna sig på en viss position, d.v.s. högst N bilar kan befinna sig i rondellen samtidigt.



FIGUR 1. *Bilarnas placering i exemplet vid tiden 0, 1, 2 och 3 för de två olika blink-scenarierna. I indatat ges vägarna i ordningen röd, blå, svart och vit.*

Under varje "tidsenhet" sker följande händelser (samtidigt):

- Varje bil som är i rondellen och som befinner sig på positionen omedelbart före sitt mål lämnar rondellen.
- Varje annan bil som är i rondellen flyttar sig en position framåt.

- Den första bilen i varje kö kör in i rondellen till positionen omedelbart efter tillfartsvägen, förutsatt att den *vet* att denna position inte kommer att upptas av en annan bil. Denna vetskap kan antingen bero på att positionen omedelbart före tillfartsvägen är tom, eller att den bil som befinner sig där just nu tänker lämna rondellen *och* visar detta genom att blinka.

På första raden i filen `rondell.dat` står ett heltal N ($3 \leq N \leq 100$), antalet tillfartsvägar. Därefter följer N par av rader, där varje par beskriver kön vid en tillfartsväg. Köerna anges i moturs ordning. På första raden i varje par står ett heltal K ($1 \leq K \leq 100$), antalet bilar i kön. På andra raden står K heltal mellan 1 och $N-1$. Dessa tal anger hur många steg framåt (moturs) som varje bil ämnar köra. Talens ordning är samma som bilarnas, det första talet gäller alltså den bil som kör in först i rondellen o.s.v.

Programmet ska skriva ut efter hur många tidsenheter som alla bilarna har kört igenom rondellen så att den åter är tom. Detta ska göras för vart och ett av de två blinkscenarierna ovan. Notera att tidsåtgången utan blink aldrig kan vara lägre än med blink.

Körningsexempel: Filen `rondell.dat` med innehållet

```
4
5
1 2 3 2 3
6
2 3 1 3 2 3
5
2 3 1 1 3
5
1 1 3 2 2
```

ska ge svaret

```
Med blink: 14
Utan blink: 19
```

UPPGIFT 3 – COLAAUTOMAT

På mitt jobb finns en colaautomat där en burk Coca-Cola kostar 8 kr. Maskinen har ett myntinkast som accepterar 1 kr, 5 kr och 10 kronorsmynt. Efter att tillräckligt många mynt har stoppats in kan jag trycka på Cola-knappen och få ut en Cola och eventuell växel. Växeln ges alltid tillbaka med det minsta antalet möjliga mynt. Denna procedur upprepas tills jag köpt så många Cola-burkar som jag vill ha. Jag kan bara köpa en burk åt gången. Om jag t.ex. skulle stoppa in 16 kr och trycka på Cola-knappen skulle jag få ut en burk och 8 kronor i växel (en 5-krona och 3 enkronor).

Givet hur många burkar jag vill köpa, samt hur många mynt jag har av varje valör, vad är det minsta antalet mynt jag behöver stoppa in i maskinen för att köpa så många burkar som jag vill ha? Jag kan ta upp mynt som kommer ut som växel. Du kan utgå ifrån att maskinen har obegränsat med mynt i växel.

Programmet ska, i tur och ordning, fråga efter hur många Cola-burkar jag vill köpa (mellan 1 och 40), antalet 1-kronors mynt jag har (mellan 0 och 100), antalet 5-kronors mynt (mellan 0 och 20), och antalet 10-kronors mynt (mellan 0 och 10). Programmet ska skriva ut det minsta antal mynt jag behöver stoppa i myntinkastet. Du kan utgå ifrån att jag har tillräckligt med pengar för att köpa alla Cola-burkar jag vill ha.

Körningsexempel:

```
Antal burkar ? 2
Antal 1-kronor ? 2
Antal 5-kronor ? 1
Antal 10-kronor ? 1

Du behöver stoppa in 5 mynt.
```

Förklaring: Först stoppar jag in en 10-krona och trycker på Cola-knappen. Ut kommer första burken och två enkronor. Därefter stoppar jag in 5-kronan och tre enkronor (en av dessa kommer från föregående växel). Totalt måste jag stoppa in 5 mynt.

UPPGIFT 4 – KORTBLANDARE

En kortblandare är en maskin som blandar en kortlek slumpmässigt. En dålig (men tyvärr vanlig) design av en sådan maskin fungerar som följer: Maskinen slumpar fram ett heltal M mellan 1 och MAX. Därefter utförs M stycken identiska blandoperationer ("rifflingar"). En blandoperation innebär att korten permuteras (kastas om) på ett visst sätt. Permutationen är kodad i maskinen och kan inte ändras. Efter att de M blandoperationerna utförts delas korten ut till spelarna. Varje spelare får K stycken kort.

En ohederlig spelare har lyckats ta reda på hur maskinen fungerar, och känner således till blandpermutationen och talet MAX. Han har dessutom identifierat vilka K kort han helst vill ha. Innan kortleken läggs in i blandaren kan han ordna korten som han vill. Bestäm hur han bäst ska ordna korten för att få så många av de K korten han vill ha. Programmet ska skriva ut det antal kort han kan förvänta sig att få, i snitt, om han ordnar dem på bästa sätt.

Första raden i filen `kort.dat` innehåller tre heltal: N (antal kort i kortleken, mellan 10 och 52), talet MAX (mellan 1 och 1000) och K (mellan 1 och N). Därefter följer en rad med N heltal (mellan 1 och N) som anger maskinens blandpermutation. Det i:te talet i denna rad anger var kortet på plats i kommer att hamna efter att en blandoperation utförts. Därefter följer en rad med K heltal som anger vilka kort spelaren kommer att få, där 1 innebär att han kommer få det första kortet i leken efter att den blandats o.s.v.

Programmet ska skriva ut det förväntade antalet önskade kort som spelaren kan få om han preparerar leken på bästa sätt. Svaret ska vara korrekt med minst 3 decimalers noggrannhet.

Körningsexempel: Filen `kort.dat` med innehållet

```
10 3 4
2 3 4 5 6 7 8 9 10 1
1 3 5 6
```

ska ge svaret

```
Förväntat antal kort: 2.667
```

Förklaring: Varje blandning förskjuter korten ett steg bakåt i leken (det sista kortet placeras överst). Genom att preparera leken så att de fyra önskade korten initialt befinner sig på position 2, 3, 4 och 10 uppnås det bästa resultatet. Beroende på hur talet M väljs av maskinen kommer de önskade korten antingen hamna på position (3, 4, 5, 1), (4, 5, 6, 2) eller (5, 6, 7, 3). I snitt fås därför (3 + 2 + 3) / 3 av de önskade korten.

UPPGIFT 5 – BOKSTAVSLEK

Två personer, låt oss kalla dem Anna och Bosse, turas om att säga en bokstav (Anna säger första bokstaven, Bosse andra, Anna tredje o.s.v.). Samma bokstav kan användas flera gånger. Det finns två regler:

- De bokstäver som hittills är sagda (i ordning) får inte bilda ett avslutat ord.
- De bokstäver som hittills är sagda (i ordning) måste kunna bilda ett ord om fler bokstäver får läggas till på slutet.

Den som bryter mot någon av reglerna förlorar och den andre vinner sålunda (i den verkliga leken kan man bluffa på den andra regeln).

Exempelvis, om bokstäverna `S`, `T` och `U` är sagda, så förlorar Bosse om han säger `P` eftersom `STUP` är ett ord. Vidare förlorar han om han säger t.ex. `J` eftersom inget ord börjar på `STUJ`. Däremot går leken vidare om han säger `G`, Anna får då inte säga `A` eftersom `STUGA` är ett ord, däremot kan hon säga `B` och därmed tvinga Bosse att avsluta ordet `STUGBY`.

För enkelhets skull antar vi här att endast bokstäverna `A-Z` får användas. Vidare finns en ordlista som innehåller alla bokstavskombinationer som ska betraktas som ord i just denna lek. Skriv ett program som bestämmer vilka bokstäver som är "säkra" att börja med för Anna, d.v.s. de bokstäver som gör att hon kan vinna omgången oavsett vilka bokstäver som Bosse väljer.

På första raden i filen `bokstav.dat` står ett heltal N ($1 \leq N \leq 40000$), antal ord i ordlistan. Därefter följer N rader med ett ord på varje rad. Orden står i alfabetisk ordning. Varje ord består av högst 20 bokstäver, valda bland versalerna `A-Z`. Även påhittade ord kan förekomma.

Programmet ska skriva ut en alfabetiskt ordnad lista över de bokstäver som har egenskapen att, om Anna börjar med bokstaven kommer hon kunna vinna oavsett vilka bokstäver Bosse väljer (båda får förstås anpassa sina val efter vad den andre säger).

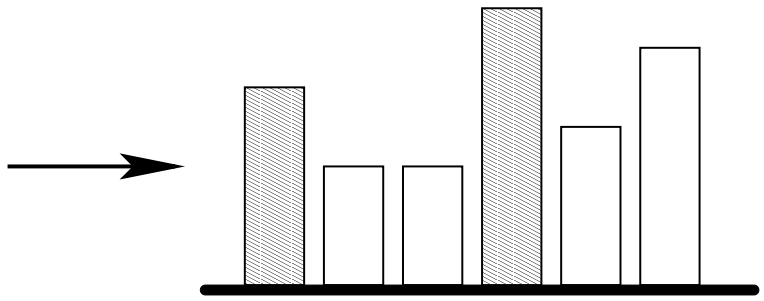
Körningsexempel: Filen `bokstav.dat` med innehållet

```
8
FE
FRI
FRIA
KO
SE
STUGA
STUGBY
STUP
```

ska ge svaret

```
Säkra bokstäver: K S
```

UPPGIFT 6 – TROFÉHYLLAN



FIGUR 2. *Troféhyllan i exemplet. Pilen visar från vilket håll hyllan betraktas. De streckade troféerna är de två som ska tas bort för att öka antalet synliga troféer från två till tre.*

Oskar har N troféer stående på en perfekt rad på sin hylla. Han bekymrar sig dock över att man inte kan se allihop när man tittar längs raden, eftersom en trofé med höjden H skymmer alla bakomvarande troféer med en höjd som är mindre eller lika med H. Oskar undrar därför om han, genom att plocka bort några troféer (men inte flytta om dem) kan göra så att fler blir synliga. Skriv ett program som, givet höjden för varje trofé i raden, beräknar hur många troféer som maximalt kan bli synliga om han plockar bort valfritt antal troféer. Programmet ska också ange vilka troféer han ska ta bort. Om det finns flera sätt att få det maximala antalet synliga troféer, ska antalet borttagna troféer vara så litet som möjligt.

På första raden i filen `trofe.dat` står ett heltal N, antalet troféer ($2 \leq N \leq 100$). Sedan följer en rad med N heltal, höjden för varje trofé ($1 \leq H \leq 100$), i den ordning de står på hyllan (den närmaste först).

Programmet ska skriva ut det maximala antalet troféer som kan synas om man får plocka bort valfritt antal. Dessutom ska programmet skriva ut en minimal uppsättning troféer som ska plockas bort för att uppnå detta. Troféerna är numrerade från 1 till N i den ordning de står på hyllan. Om det finns flera olika lösningar med minimalt antal troféer bortplockade ska programmet ange vilken som helst av dem.

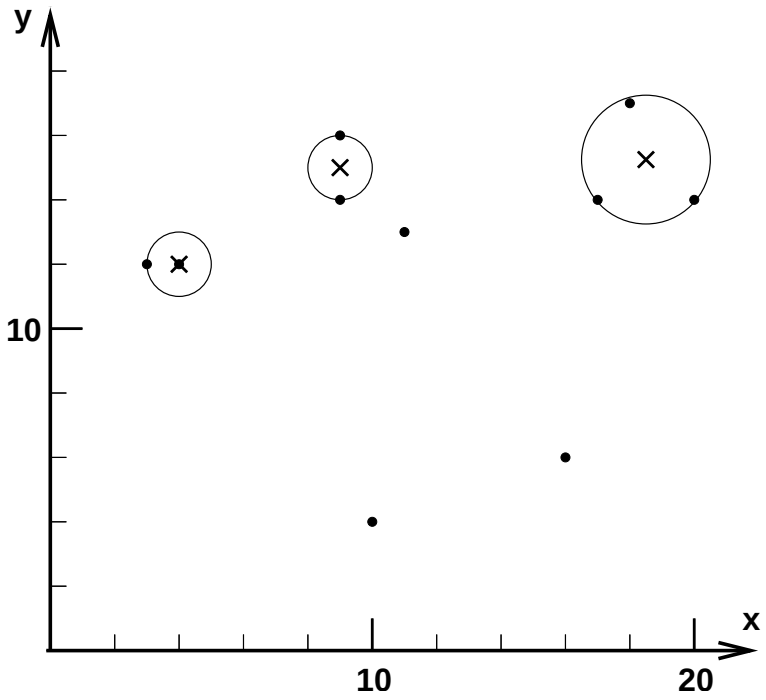
Körningsexempel: Filen `trofe.dat` med innehållet

```
6
5 3 3 7 4 6
```

ska ge svaret

```
Max synliga: 3
Ta bort: 1 4
```

UPPGIFT 7 – VAKTTORN



FIGUR 3. *Punkterna visar de viktiga platserna i exemplet och kryssen visar en möjlig placering av de tre vakttornen för att kunna bevaka sju platser. Torn 1 övervakar punkterna 4 och 10, torn 2 övervakar punkterna 7 och 8, och torn 3 övervakar punkterna 2, 5 och 6.*

Du önskar bygga tre vakttorn med varierande höjd för att övervaka ett antal viktiga platser. Skriv ett program som, givet koordinaterna på platserna du vill övervaka, samt hur långt du kan se från vart och ett av tornen, bestämmer var tornen ska placeras så att så många platser som möjligt kan övervakas från åtminstone ett torn. Tornen kan placeras varsomhelst på xy-planet, inklusive på en plats som önskas övervakas.

Rent formellt kan både platserna och tornen betraktas som punkter i xy-planet. Om det euklidiska avståndet mellan ett torn och en punkt inte överskrider tornets betraktningsavstånd kan punkten övervakas.

Första raden i filen `vakttorn.dat` innehåller en rad med ett tal N ($1 \leq N \leq 40$), antalet punkter du vill övervaka. Därefter följer en rad med tre heltal som anger betraktningsavstånden från vart och ett av tornen ($1 \leq h_i \leq 1000$). Sedan följer N rader som anger koordinaterna för var och en av punkterna du vill övervaka. Alla koordinater är heltal mellan 0 och 1000. Två punkter har aldrig samma koordinater.

Skriv ut det maximala antalet punkter som kan övervakas om tornen placeras optimalt, samt ge ett exempel på var tornen ska placeras för att realisera detta. Notera att tornen inte nödvändigtvis behöver placeras på heltalskoordinater (3 decimaler är tillräckligt i utskriften).

Körningsexempel: Filen `vakttorn.dat` med innehållet

```
10
1 1 2
10 4
18 17
16 6
9 14
17 14
20 14
4 12
3 12
11 13
9 16
```

kan ge svaret

```
Platser som kan övervakas: 7
Torn 1: 9, 15
Torn 2: 4, 12
Torn 3: 18.5, 15.25
```

Observera att det finns andra placeringar som ger samma antal övervakade platser.