

Omkontrollskrivning, Programmeringsteknik för D/C

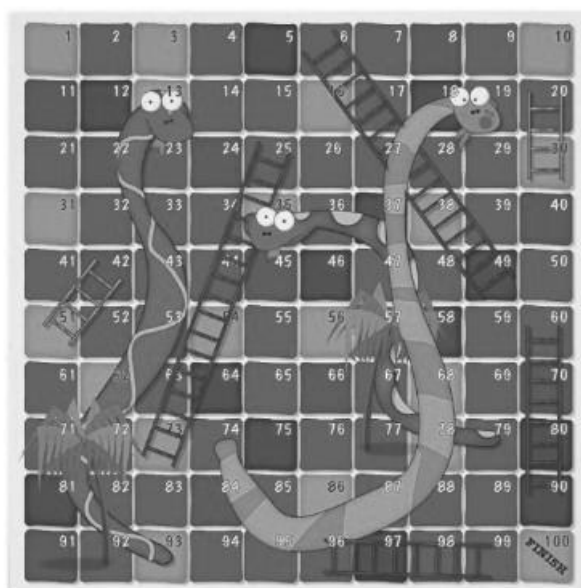
2013-04-03, 14.00–19.00

Anvisningar: Preliminärt ger uppgifterna $12 + 6 + 7 + 15 = 40$ poäng. För godkänt betyg krävs ungefär 25 poäng. Tillåtet hjälpmedel: Java-snabbreferens.

Betyget på denna skrivning utgör betyget på kontrollskrivningarna, oavsett resultatet på tidigare skrivningar (dock sänks inte ett tidigare godkänt betyg). Resultatlistan anslås på institutionens anslagstavla när rättningen är klar.

Uppgift 1–3 handlar om spelet ormar och stegar. Uppgift 4 är en helt fristående uppgift.

1. I tärningsspelet ormar och stegar används en spelplan med rutor numrerade från 1 och uppåt. På spelplanen finns ormar och stegar som leder från en ruta till en annan. Ormar leder bakåt dvs. till en ruta med lägre nummer. Stegar leder framåt dvs. till en ruta med högre nummer.



I varje drag kastar spelaren tärningen och flyttar fram sin spelpjäs så många rutor som tärningen visar. Om spelpjäsen då hamnar på en ruta som är startruta för en orm eller stega flyttas spelpjäsen direkt vidare till den ruta som ormen eller stegen leder till. När man hamnat på eller passerat sista rutan har man gått i mål.

Ytterligare regler:

- I en ruta kan det högst börja en orm eller stega. Däremot är det möjligt att flera ormar/stegar slutar i samma ruta.
- Det är även möjligt att en ruta som avslutar en orm/stega innehåller början på en annan orm/stega. Om man blir förflyttad till en sådan ruta via en orm/stega så ska man stanna på den rutan och inte följa den nya ormen/stegen.
- Första rutan (nr 1) och sista rutan innehåller inga ormar eller stegar.

Exempel på en spelplan med 100 rutor, 3 ormar och 6 stegar (samma exempel som på bilden):

```
1 2 3 4 5 49 7 8 9 10 11 12 13 14 15 16 17 18 19 30 21 22 23 24 73 26 27 28 29 30 31 32 23 34 35 36 37 38
39 40 41 51 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 90 61 62 63 64 65 66 67 68 69 70 71 72 73
19 75 76 77 78 35 80 81 82 83 84 85 86 87 88 89 90 91 92 13 94 95 99 97 98 99 100
```

Till exempel innehåller ruta 6 en stega som slutar på ruta 49.

Klassen GameBoard beskriver en spelplan:

GameBoard

```
/** Skapar en spelplan med nbrSquares rutor, numrerade från 1 (startrutan)
    till nbrSquares (målrutan). Slumpar ut nbrSnakes ormar och nbrLadders
    stegar på spelplanen enligt spelets regler. */
GameBoard(int nbrSquares, int nbrSnakes, int nbrLadders);

/** Returnerar numret på spelplanens startruta. */
int getStart();

/** Returnerar numret på spelplanens målruta */
int getFinish();

/** Returnerar numret på den ruta ormen eller stegen som börjar i rutan nbr
    leder till. Om det inte finns någon orm eller stega som börjar i nbr
    returneras nbr. */
int getToSquare(int nbr);

/** Skriver ut alla ormar och stegar dvs. rutan som ormen eller stegen börjar
    på och rutan ormen eller stegen slutar på. */
void printSnakesAndLadders();
```

Implementera klassen GameBoard. Ledning och anvisningar:

- Låt spelplanen representeras av en heltalsvektor.

```
int[] toSquare
```

där toSquare[nbr] för rutan nbr anger vart ormen eller stegen som börjar i denna ruta leder. Vektorelementet med index 0 används inte. För de rutor som inte är startpunkt för någon orm eller stega så innehåller toSquare[nbr] numret på rutan själv, dvs nbr.

- Utskriften i metoden printSnakesAndLadders ska se ut enligt följande exempel (samma exempel som ovan):

```
6 49
20 30
25 73
42 51
60 90
74 19
79 35
93 13
96 99
```

2. Normalt är man två eller flera som spelar ormar och stegar och som omväxlande gör sitt drag. Men i den här uppgiften vill vi istället beräkna statistik över hur många drag det i genomsnitt krävs för att komma från start till mål. Statistiken kan användas för att värdera om en spelplan fungerar bra eller ej.

Vi behöver därför en klass med en metod som låter en spelare göra alla sina drag från start till mål:

SolitairePlay

```
/** Skapar ett objekt som låter en deltagare spela ormar och stegar. */  
SolitairePlay(GameBoard board, Die d);  
  
/** Spelar en omgång ormar och stegar med endast en deltagare. Antalet drag  
    returneras. */  
int playOneSolitaireRound();
```

Den färdiga klassen Die beskriver en tärning och har följande specifikation:

Die

```
/** Skapar en tärning. */  
Die();  
  
/** Kastar tärningen. */  
void roll();  
  
/** Returnerar antal prickar tärningen visar. */  
int getResult();
```

Implementera klassen SolitairePlay.

3. Skriv ett program som genomför ett stort antal spelomgångar av spelet ormar och stegar. Antalet spelomgångar ska läsas in. I slutet av programmet ska följande skrivas ut:
- spelplanens ormar och stegar,
 - antalet kast per spelomgång i genomsnitt,
 - antalet kast i den spelomgång som krävde flest kast.

4. I många datorspel håller man reda på spelarnas resultat i en highscorelista. Den färdiga klassen `Player` beskriver en spelare och innehåller spelarens namn och hittills bästa resultat (poäng):

`Player`

```
/** Skapar en spelare med namnet name och noll poäng. */
Player(String name);

/** Tar reda på spelarens namn. */
String getName();

/** Tar reda på spelarens bästa resultat. */
int getMaxPoints();

/** Uppdaterar spelarens bästa resultat om points är bättre än det hittills
    bästa resultatet. Returnerar true om resultatet har uppdaterats, annars
    false. */
boolean setMaxPoints(int points);
```

Klassen `HighScoreList` innehåller en lista med de spelare som har ett resultat. Om en spelare får ett nytt resultat ska det ersätta det gamla om det nya resultatet är bättre.

`HighScoreList`

```
/** Skapar en tom highscorelista. */
HighScoreList();

/** Spelaren name har fått ett nytt resultat. Uppdatera spelarens resultat
    om det nya resultatet är bättre än det gamla. Om spelaren inte finns i
    listan ska spelaren läggas in. Returnerar true om listan uppdaterats
    (spelaren fick ny högre poäng eller var ny), i annat fall false. */
boolean update(String name, int points);

/** Returnerar en lista med de maxNbr spelare som har högst poäng.
    Om antalet spelare i listan understiger maxNbr så returneras en lista
    med alla spelare. */
ArrayList<Player> getBest(int maxNbr);
```

Implementera klassen `HighScoreList`. Ledning och anvisningar:

- Håll highscorelistan sorterad med avseende på avtagande poäng. Förutom de publika metoderna i specifikationen *ska* du också lägga till och som hjälp använda följande två privata metoder:

```
/* Returnerar index i listan för spelaren med namnet name. Om spelaren
    inte finns i listan returneras -1. */
private int find(String name) {...}

/* Läger in personen p på rätt plats i listan. Håller listan sorterad
    efter avtagande poäng. */
private void add(Player p) {...}
```

- Det kan finnas flera spelare med samma poäng. Därför kan det returneras en lista i `getBest` där de sista spelarna har samma poäng som spelare som hamnar utanför denna lista. Detta är ok.