Kontrollskrivning 2, Programmeringsteknik för D/C

2012–11–19, 13.15–16.15

Anvisningar: fyll i omslaget fullständigt, även lösta uppgifter och antalet inlämnade blad. Skriv dina lösningar på rutpapper, bara på en sida av varje papper. Lämna bara in dina lösningar och omslaget, inte skrivningen och inte några kladdpapper. Tillåtet hjälpmedel: Java-snabbreferens.

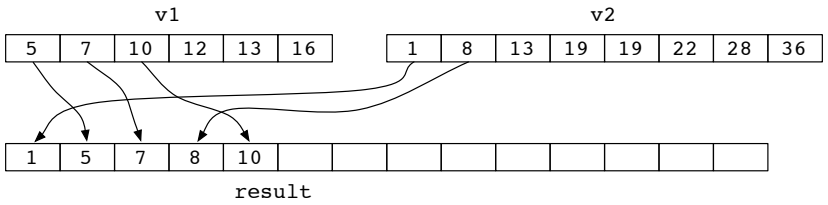
Preliminärt ger uppgifterna $15 + 15 + 10 = 40$ poäng.

Vi meddelar på kurshemsidan när rättningen är klar och sätter upp en lista med poäng på anslagstavlan. Senare blir det visning av skrivningen.

---

1. Samsortering (”merge” på engelska) kallas det när man har två sorterade följder och man slår ihop dem till en ny sorterad följd. Vi förutsätter här att följderna består av heltal som är lagrade i två vektorer `v1` och `v2`. När man ska samsortera vektorerna till en ny vektor `result` så jämför man hela tiden de första ”ej avklarade” talen i `v1` och `v2` och lägger det minsta av dessa tal i `result`. Detta upprepas tills alla talen i båda vektorerna är avklarade.

   Så här kan det se ut när man har klarat av tre tal från `v1` och två tal från `v2`:



   Man började med att jämföra 5 och 1: 1 var minst och lagrades i `result`. Sedan jämförde man 5 och 8: 5 var minst och lagrades i `result`. 7 och 8: 7 var minst och lagrades i `result`, och så vidare. Observera att det kommer att finnas tal kvar i `v2` när alla talen i `v1` är avklarade. Dessa tal ska flyttas till `result`.

   Skriv en metod som utför samsorteringen. Metoden ska ha följande rubrik:

```
public static int[] merge(int[] v1, int[] v2);
```

   Metoden ska alltså returnera den samsorterade vektorn. Observera att `v1.length` eller `v2.length` kan vara $= 0$ — metoden ska fungera också i detta fall.

   Tips: inför tre heltalsvariabler `i`, `j` och `k`. `i` ska hålla reda på index för det första ej avklarade talet i `v1`, `j` ska göra samma sak i `v2`, och `k` ska hålla reda på den första lediga platsen i `result`.

2. En student vid en högskola beskrivs av följande klass (vi använder bara en av metoderna i klassen):

Student

```
/** Tar reda på antalet poäng som studenten har klarat under året year */
int getPoints(int year);
```

Alla studenterna på högskolan beskrivs av ett objekt av följande klass:

```
public class StudentRegister {
    private int nbrStudents;    // antalet studenter
    private Student[] students; // vektor med nbrStudents studenter

    // konstruktorer och metoder där vektorn students bildas

    /** Beräknar medelantalet poäng som studenterna klarade under året year */
    public double getAveragePoints(int year) {
        ...
    }

    /** Räknar hur många studenter som under året year klarade  0-9, 10-19, ...,
        50-59, >=60 poäng, skriver ut antalen med ett tal per rad */
    public void printStatistics(int year) {
        ...
    }
}
```

Implementera metoderna `getAveragePoints` och `printStatistics`.

3. En klass som håller reda på heltal har följande utseende:

```
public class IntList {
    private ArrayList<Integer> list;

    /** Skapar en tom heltalslista */
    public IntList() {
        list = new ArrayList<Integer>();
    }

    /** Lägger in talet nbr sist i listan (efter de tal som redan finns) */
    public void add(int nbr) {
        ...
    }

    /** Tar bort de tal i listan vars index finns i listan toRemove (se
        exempel nedan). Talen i toRemove är sorterade i växande ordning
        och ligger inom gränserna för listan  */
    public void remove(ArrayList<Integer> toRemove) {
        ...
    }
}
```

Exempel på hur `remove` ska fungera:

```
list före remove:   3 1 66 3 4 -1 51
toRemove:           2 3 6
list efter remove:  3 1 4 -1
```

Implementera metoderna `add` och `remove`.