Data och Informationsteknik / Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
Erland Holmström Göteborg 26 nov 2013

Övning 4

Läs igenom övningen innan övningstillfället och fundera över vad du behöver lära dig (och lär dig det). Det är ok att börja fundera på uppgifterna hemma.

De rekursiva program vi skriver nu kan lika gärna skrivas iterativt, det blir inte så stor skillnad i längd eller svårighetsgrad. När man behandlar datastrukturer som träd är rekursion nästan nödvändig. (Nästan, eftersom alla rekursiva program kan skrivas iterativt men ibland mycket svårare och obegripligare.)
Främsta nyttan med rekursion är dock som tankesätt och problemlösningsmetod. En rekursiv lösning i Java verkar vara ca 3 gånger så långsam som en iterativ. Men detta uppvägs mer än väl av rekursions kraftfullhet som problemlösningsmetod, det mesta av koden vi skriver är ju inte tidskritisk.

Övningar

1. Vad skrivs ut om `count1` respektive `count2` anropas med argumentet 5? Handexekvera beräkningen med papper och penna! (Innan du provkör.) Det är viktigt att du förstår varför resultatet blir som det blir.

a)
```
static void count1(int n) {
    if (n > 0) {
      System.out.print(n + " ");
      count1(n-1);
    }
  }
```

b)
```
static void count2(int n) {
    if (n > 0) {
      count2(n-1);
      System.out.print(n + " ");
    }
  }
```

2. Skriv en rekursiv funktion, `int numberOfDigits(int n)` som beräknar hur många siffror det är i ett givet naturligt tal. Inledande nollor räknas ej utom för talet 0 som skall ge svaret 1. Exempel:

`println(numberOfDigits(-12345))` skriver ut 5

`println(numberOfDigits(3723400))` skriver ut 7

`println(numberOfDigits(0))` skriver ut 1

OBS: Java tolkar tal med inledande nollor som oktala tal (dvs som angivna i basen 8) så `numberOfDigits(00010)` returnerar 1 :-(Ert program behöver inte hantera tal med inledande nollor förutom talet noll. Detta gäller när man tilldelar eller skriver in talet i direkt i koden.

3. Skriv en rekursiv metod som tar en sträng som parameter och returnerar samma sträng men med allt utom bokstäverna borttagna.

```
String strip(String str)
```

Ex: `strip("abc123,.-de f")` -> "abcdef". Du kan använda metoder i klasserna String och Character för att ta ut delsträngar mm. `isLetter` är också en användbar metod.

4. Om man vill integrera en "snäll" funktion, fun, kan man använda följande algoritm: Om intervallet man skall integrera över är litet (mindre än en given konstant eps, säg 10^{-6}) kan man få en bra approximation genom att multiplicera intervallängden med funktionen fun's värde i intervallets mittpunkt. Om intervallet är större kan man dela det i två lika delar och summera integralerna av vardera delen.

a)

Skriv en rekursiv funktion integrate som integrerar en funktion fun som är deklarerad i samma klass. Integrate har som argument även ändpunkterna i det intervall över vilket man vill integrera dvs (antag att a<=b)

```
static double integrate(double a, double b, double eps)
```

Skriv sedan ett huvudprogram (dvs en main) som integrerar funktionen $fun(x) = x^2$ över intervallet 0..1.

Man skall kunna integrera olika funktioner utan att behöva ändra koden i metoden `integrate` dvs deklarera funktionen fun som en egen metod i samma klass som `integrate`. Deklarera eps som en konstant. Funktionerna som skall integreras kan antas ha positiva funktionsvärden.

b)

Lösningen ovan är naturlig för en nybörjare. Man deklarerar funktionen att integrera (FAI) inuti klassen som integrerar. Nackdelen med detta är att man måste ändra i klassen som innehåller integrera varje gång man vill byta FAI. Istället borde man försöka abstrahera ut FAI och ha den som parameter till den integrerande funktionen enligt

```
static double integrate(Function fai, double a, double b, double eps)
```

men man kan inte ha funktioner som parameter till en metod. Däremot kan man ha ett objekt som parameter (som tex kan heta Function) och ett objekt kan innehålla en funktion (dvs en metod).

Låt klassen som implementerar FAI implementera interfacet nedan och lös uppgiften på detta sätt.

```
public interface Function {
  public double fun(double x);
}
```

5. Skriv en *rekursiv* funktion

```
public static int[] fromTo(int a, int b)
```

som givet två heltal a och b returnerar en vektor med innehåll (a, a+1, a+2, …, b). Om a>b returneras en tom vektor.

Den här typen av rekursiva funktioner är ganska knöliga att skriva i Java, du behöver en wrapper.

6. Definiera ett interface Filter enligt:

```
public interface Filter {
  public boolean accept(Object obj);
}
```

Skriv sedan en metod

```
public static ArrayList filter(ArrayList al, Filter f)
```

som returnerar en lista med alla element i `al` som accepterades av filtret `f`.

Skriv sedan en klass som implementerar Filter och som filtrerar ut alla tal > 0 (av typ Integer eller Double). Och en annan som filtrerar ut alla strängar som är minst 3 tecken långa. (För varje specifikt villkor skriver man alltså en subklass till Filter)
Avsluta med ett huvudprogram som testar.

7. Delmängdssummeproblemet - SOS. En lite svårare uppgift.

Kan man ur en grupp med föremål $F_1..F_n$ med vikter $W_1..W_n$ välja en delmängd som väger exakt M kilo? Algoritmen kan tex användas för att packa flygplan, sateliter mm så optimalt som möjligt.

Ex: Föremål:	F_1	F_2	F_3	F_4	F_5
Vikt:	3	7	2	3	6

Här kan vi packa alla vikter upp till 21 utom 1, 4, 17, 20. Tex F_2, F_3 och F_5 ger 15 kilo.

Algoritm idé: pröva alla alternativ - två fall

1) **Tag med** det sista föremålet och försök packa resterande utrymme med $F_1..F_{n-1}$.

2) **Tag inte med** det sista föremålet och försök packa resterande utrymme med $F_1..F_{n-1}$.

Man kan naturligtvis också börja med det första föremålet.

Definiera predikatet `packa(W, M, i)` som är sant om det finns en delvektor i $W_1..W_i$ vars summa är M och falsk annars. Fallanalys: (tre basfall, två rekursionssteg)

1) Om $M = 0$ så har vi lyckats.

2) Om $M < 0$ (sista föremålet var för stort) så har vi misslyckats.

3) Om $i = 0$ (föremålen är slut trots att det finns plats kvar) så har vi misslyckats.

4) Tag med det sista föremålet och försök packa resterande utrymme med $F_1..F_{n-1}$.

5) Tag inte med det sista föremålet och försök packa resterande utrymme med $F_1..F_{n-1}$.

Rekursionsantagande: vi antar att `packa(W,M,i-1)` löser problemet för föremålen $F_1..F_{i-1}$.