Rättningsmall i DD1361 Programmeringsparadigm, 29 november 2008

Kursansvarig: Lars Arvestad

- Denna rättningsmall innehåller vägledning i rättningen. När du är lite osäker över hur du ska sätta poäng bör du först och främst använda ditt eget omdöme! Det brukar vara bra.
- Kom ihåg att det är tentandens uppgift att övertyga dig om att lösningen är korrekt. Du ska inte behöva anstränga dig för att förstå lösningen.
- Om du är mycket osäker på hur du ska rätta kan du göra en markering om det på försättsbladet och så gör Lasse en bedömning.
- Glöm inte att föra över poängen till försättsbladet! Var snäll och summera poängen!

Allmänna frågor

1. Några exempel är C, Pascal, Java, Python, Ada, Modula 3, Simula, C#, men andra språk är också godtagbara. Ett poäng per språk. (2p)
2. Några exempel är C, C++, Fortran, Algol, Pascal. Ett poäng per språk. (2p)
3. Överlagring innebär att man definierar samma funktion med olika typsignaturer. Man kan till exempel tänka sig en funktion `append` som är definierad på både heltalslistor, strängar, och listor med strängar. (2p)

   Denna uppgift genererade en hel del reaktioner, så jag har kollat upp litteraturen och sökt på nätet för att se om ordet ”överlagring” är omdiskuterat. Jag har inte funnit tecken på det.

   På tentan har ett flertal blandat ihop överlagring (overloading) med omdefinition (redefinition eller overriding) då en ärvd metod, med samma typsignatur som tidigare, får en ny implementation. Det verkar även kallas ”överskuggning” på svenska.

   Litteraturen som jag tittat i:

   - Sebesta, ”Concepts of Programming Languages”.
   - Barnes och Kölling, ”Objects first with Java”.
   - Jan Skansholm, ”Java direkt med Swing”.
4. Multipelt arv innebär att du kan skriva klasser som har flera superklasser, dvs en klass A kan ärva kod och egenskaper från klass B och C. (2p)
5. Att ändra på en global variabel, att läsa och skriva till en buffert/fil, att ändra på ett attribut i ett objekt, mm. För full poäng ska det vara ett exempel. Dra ett poäng om det är en allmän förklaring, tex “ändrar tillståndet på programmet”. (2p)
6. Currying. (2p)

Funktionell programmering i Haskell

7. (a) `map length` har typen `[[a]] -> [Int]`, eftersom `map` använder funktionen `length` på alla element i listan som ges som andra argument, men är utelämnad här. (2p)

   (b) `[(x, odd x) | x <- [1..]]` har typen `Integral a => [(a, Bool)]`. Jag ger full poäng även för `[(Int, Bool)]`. (2p)

8. En möjlig lösning är:

```
and :: [Bool] -> Bool
and [] = True
and (b:bs) = if b then
               and bs
             else
               False
```

   En snyggare lösning är

```
and :: [Bool] -> Bool
and [] = True
and (True:bs) = and bs
and (False:bs) = False
```

   Dra av en poäng om basfallet saknas eller är fel. (3p)

9. Beroende på hur man definierar sin funktion kan typsignaturen bli olika. Eftersom C-funktionen var definierad över `int` så är det rimligt att skriva

```
[Int] -> [Int] -> Int
```

   men alternativen

```
Num a => [a] -> [a] -> a
```

```
Integral a => [a] -> [a] -> a
```

```
[Integer] -> [Integer] -> Integer
```

   är också godtagbara. Typsignaturen ger 1 poäng.

   Det finns förstås många varianter på `inreprodukt`, till exempel

```
inreprodukt v1 v2 = sum (zipWith (*) v1 v2)
```

   eller lätt modifierat:

```
inreprodukt v1 v2 = sum $ zipWith (*) v1 v2
```

   Den renlärige rekursionsprogrammeraren kanske hellre skriver:

```
inreprodukt [] [] = 0
inreprodukt (x1:v1) (x2:v2) = x1 * x2 + ip v1 v2
```

   Brist på basfall ger en poängs avdrag. (4p)

10. Man ska se att p1 är en konjunktion av predikaten p2 och p3 (2 p). Det måste finnas en regel eller faktum som gör att p2 lyckas (1 p) och en regel eller faktum som gör att p3 är *definierad* men ändå misslyckas (1 p). Ett enkelt exempel:

```
p1(X) :- p2(X), p3(X).
p2(piff).
p3(puff).
```

Och ett alternativ:

```
p1(X) :- p2(X), p3(X).
p2(X) :- true.
p3(X) :- fail.
```

11. - Predikatet `s` beräknar "tecknet" på första parametern (1 p)
    - och `sv` använder detta predikat över hela lista i första parametern och returnerar svaret i andra parametern (2 p).

12. Den enklaste lösningen är

```
utan(X,L) :- length(L, X).
```

dvs man kör `length` "baklänges". En annan lösning är att återimplementera `length`:

```
utan(0, []).
utan(K, [_|L]) :-
        K2 is K - 1,
        utan(K2, L).
```

Att explicit skapa listor av icke-unifierade variabler, tex

```
utan(0, []).
utan(1, [_]).
utan(2, [_, _]).
...
```

är inte en godtagbar lösning. (2p)

Syntaxanalys

13. (a) En enkel lösning är:

`(triathlon)|(quadrathlon)|(pentathlon)`

För att spara bokstäver kan man också tänka sig:

`(tri)|(quadr)|(pent)athlon`

En lösning som känner igen *fler* ord är inte godtagbar. (2p)

(b) Förslag: `[A-Z]([A-Z][A-Z])*`. (2p)

14. (a)
- "Jag öh sprang snabbt typ" (1p)
- "Jag öh typ öh sprang snabbt" (1p)

(b) Rot-produktionen har tre subjekt och tre pauser att välja mellan, samt antalet aktiviteter. Antalet aktiviteter kommer från två olika meningstrukturer.

Den första strukturen har $3 \times 2 \times 2 = 12$ kombinationer tack vare de olika orden man kan välja. Den andra meningstrukturen har $2 \times 3 \times 3 \times 2 = 36$ kombinationer. Detta ger att antalet meningar som grammatiken kan generera är $3 \times 3 \times (12 + 36) = 432$.

Man får gärna svara med en multiplikation här, och också förenkla till "9×48". Jag skulle inte ens dra poäng om man hade räknat fel efter en korrekt uppställning. (2p)

Denna uppgift hade fel svar i första versionen av rättningsmallen!