Tentamen i DD1361 Programmeringsparadigm, 2 juni 2009

Kursansvarig: Lars Arvestad

- Detta är en **kamraträttad tenta**! Skrivtiden är 8 - 10.50 och rättningstiden är 11 – 12.
- Du ska ha **två tentaomslag**: ett anonymt och ett slutgiltigt. På båda omslagen och alla papper ska du en kod som består av sal och platsnummer. **Om du sitter på plats 17 i sal Q21 kan du märka dina papper med "Q21:17".**
- Det anonyma omslaget ska bara ha din kod, inte ditt namn.
- Det slutgiltiga omslaget ska ha ditt namn och din kod. Detta används när tentan lämnas till studentexpeditionen.
- **Skriv inte namn** på skrivpappren. Tentorna är anonyma! Istället skriver du sal och platsnummer.
- Lämna in tentan i det anonyma omslaget när du är klar och lämna sedan salen. Återvänd till *din* plats kl 11.00 för gemensam rättning! Varje tentand rättar då en annan tentands tenta med hjälp av rättningsmall. Rättningen kommer att granskas av Lars.
- För varje baslabb du har redovisat ges 1 bonuspoäng. För godkänt på tentamen krävs 20 poäng. Möjlighet till komplettering garanteras vid 18 poäng. Observera att tentamen inte betygsätts.
- **Ni som gick kursen innan avsnittet om syntaxanalys tillkom behöver inte göra sista avsnittet på tentan.** Er tenta omfattar alltså 32 poäng och för godkänt krävs 16 poäng. Komplettering garanteras vid 14 poäng.
- **Godkända hjälpmedel:** Skrivmedel. "Prolog programming" av Brna. "Programming in Haskell" av Hutton, eller det tidigare särtrycket av samma bok. "Functional Programming" av Jeroen Fokker och "An Introduction to Computing with Haskell" av Manuel Chakravarty går också bra.

Lycka till!

Allmänna frågor

1. Vad är skillnaden mellan överlagring och omdefinition inom objektorientering? (2p)
2. Vad är dynamisk typning? Ge ett exempel på ett (känt...) programmeringsspråk med dynamisk typning. (2p)
3. Nämn två exempel på uttryckssätt som du måste undvika om du vill programmera *rent funktionellt* i språket C. (2p)
4. Vid det här laget vet du vad *polymorfiska typer* innebär i Haskell. Kan man säga att språk som Java och C++ har också har polymorfiska typer? Motivera! (2p)
5. Klassificera följande språk enligt vilka paradigmer de anses tillhöra: OO, imperativt, funktionellt, eller logikprogrammering.

   Språken: Fortran, Pascal, Python, och Lisp (2p)

Funktionell programmering i Haskell

6. Ange typsignatur för följande Haskell-uttryck.

   (a) `\x -> (x, odd x)` (2p)

   (b) `fst . snd` (2p)

7. Skriv en funktion `firstGood` som tar ett predikat (en funktion som returnerar ett värde av typen `Bool`) och en lista, och returnerar de första elementet i listan som predikatet är sant för. Returvärden ska ges med hjälp av `Maybe`-monaden. (2p)

   Exempel:

```
Hugs> firstGood odd []
Nothing
Hugs> firstGood odd [2,4,6]
Nothing
Hugs> firstGood odd [2,4,6,7]
Just 7
Hugs> firstGood odd [2,4,6,7,9]
Just 7
```

8. Skriv en evaluator för enkla aritmetiska uttryck i postfix-notation (*à la* HP-räknare). Du ska utgå ifrån att uttrycken ges som en lista av element av typen `ExprElem`:

```
data ExprElem = C Float
              | Neg
              | Add
              | Mul
```

   Det ska utgå ett fel (med hjälp av funktionen `error`) när det inte finns tillräckligt med data för operatorerna. (4p)

   Exempelkörning:

```
Hugs> eval [C 7]
7.0
Hugs> eval [C 7, C 3, Add]
10.0
Hugs> eval [C 7, C 3, Mul]
21.0
Hugs> eval [C 7, C 3, Neg, Mul]
-21.0
Hugs> eval [C 7, C 3, Mul, Neg]
-21.0
Hugs> eval [C 3, C 3, Add, C 2, C 3, Add, Mul]
30.0
Hugs> eval [Add]
Error: Empty stack.

Hugs> eval [C 1, Add]
Error: Too few operands on stack.
```

Logikprogrammering med Prolog

9. Visa med boxmodellen hur exekveringen av Prologkoden

```
ekorre(piff).
ekorre(puff).

ekorrpar(X, Y) :- ekorre(X), ekorre(Y), \+ X= Y.
```

görs i följande beräkning:

```
| ?- ekorrpar(X,Y).
X = piff,
Y = puff ?
yes
```

(2p)

10. Skriv ett Prolog-program som letar upp *perfekta tal*, dvs tal vars heltalsdelare summerar till samma tal. Exempelvis är 6 perfekt eftersom dess faktorer 1, 2 och 3 summerar till 6. Likaså är 28 perfekt eftersom det har faktorerna 1, 2, 4, 7, och 14 som summerar till 28.

    Din lösning ska använda generera-och-testa-tekniken och bestå av deluppgifterna (a) till (f) nedan.

    Tips 1: Se sidan 71 i Brnas bok.

    Tips 2: Heltalsdivision görs med `//` och resten ges med `rem`. Ex: `X is 7 rem 3` ger X = 1.

    (a) Skriv predikatet `summa` som tar en lista av tal och räknar ut deras summa. (1p)

    (b) Skriv predikatet `delare` som tar två heltal X och D som parametrar och avgör om D är en delare till X. (1p)

    (c) Skriv ett predikat `intervall` som genererar alla heltal mellan 1 och ett maximum M som ges som en parameter till predikatet. (2p)

    (d) Skriv predikatet `faktorer` som tar ett tal och beräknar alla dess faktorer. Ex: (3p)

    ```
    | ?- faktorer(6, L).
    L = [1,2,3] ? n
    no
    ```

    (e) Skriv ett predikat `perfekt` som testar om ett tal är perfekt och dessutom returnerar en lista på dess faktorer. (1p)

    (f) Skriv ett predikat `perfekta_tal` som genererar alla perfekta tal och dess faktorer. Det ska kunna ge en körning som liknar den här:

    ```
    | ?- perfekta_tal(X, L).
    L = [3,2,1],
    X = 6 ? n
    L = [14,7,4,2,1],
    X = 28 ? n
    L = [248,124,62,31,16,8,4,2,1],
    X = 496 ?
    yes
    ```

    (2p)

Syntaxanalys

Du som följde kursen innan 2005 struntar i detta avsnitt!

11. (a) Skriv ett regex som beskriver CSC:s kurskoder, dvs först två bokstäver (DD, DM, DH, DN, DT, och DA), följt av fyra siffror (generalisera till alla tänkbara kurskoder i dagens system). Till exempel är DD1361 koden för denna kurs. (1p)

    (b) Skriv ett reguljärt uttryck som matchar sig självt. Dvs, det ska beskriva ett språk som innehåller det reguljära uttrycket självt. (1p)

    (c) Skriv ett regex som beskriver sökvägar i Unix (förenklat):
    - Roten är en sökväg.
    - En sökväg kan starta vid roten eller vara relativt aktuell sökväg.
    - Fil- och katalognamn innehåller bokstäver 'a' till 'z' (och med stora bokstäver), understreck (_), och punkter.

    (2p)

12. Här nedan ser du Haskell-kod för att verifiera att en lista av element kommer från ett visst språk. Det är rekursiv medåkning baserat på en noggrann prediktiv grammatik. Härled hur grammatiken antagligen såg ut! (4p)

```
module Parsning where

data BirdSounds = Tjirp | Oirp | Oiiirp | Oarp
                | Kvidde | Vidde | Vitt | Pause
                  deriving (Eq, Show)

parse_birds :: [BirdSounds] -> (Bool, [BirdSounds])
parse_birds sounds =
    let (stat, rem) = birdsound sounds
    in if (stat == True)
       then more_sounds rem
       else (False, rem)

more_sounds :: [BirdSounds] -> (Bool, [BirdSounds])
more_sounds [] = (True, [])
more_sounds (Pause : rest) = parse_birds rest

birdsound :: [BirdSounds] -> (Bool, [BirdSounds])
birdsound (x:xs) = if (x == Tjirp) then more_tjirp xs
                   else if (x == Oirp) then more_oirp xs
                   else if (x == Kvidde) then vidde_osv xs
                   else (False, (x:xs))

more_tjirp :: [BirdSounds] -> (Bool, [BirdSounds])
more_tjirp (Tjirp : xs) = (True, xs)
more_tjirp (Pause : xs) = more_tjirp xs
more_tjirp sounds = (False, sounds)

more_oirp :: [BirdSounds] -> (Bool, [BirdSounds])
more_oirp (Oiiirp : xs) = (True, xs)
more_oirp (Oarp : xs) = (True, xs)
more_oirp sounds = (False, sounds)

vidde_osv :: [BirdSounds] -> (Bool, [BirdSounds])
vidde_osv (Vidde : Vitt : xs) = (True, xs)
vidde_osv sounds = (False, sounds)
```