Tentamen i DD1361 Programmeringsparadigm, 17 december 2010

Kursansvarig: Lars Arvestad

- Detta är en **kamraträttad tenta**! Skrivtiden är 9.00 till 12.00 och rättningstiden är 12.15 till ca 13.15.

- Tentan är anonym! På omslaget och alla papper ska du en kod som består av sal och platsnummer. **Om du sitter på plats 17 i sal Q31 ska du märka dina papper med "Q31:17".** Platsnumret ges av tentavakten.

- Du ska även skriva din kod och ditt namn på en platslista som tentavakten har så att vi kan koppla ihop tentan med ditt namn!

- **Skriv inte namn** på skrivpappren. Tentorna är anonyma!

- Vi samlas sedan **kl 12.15 i sal K1 för gemensam rättning**! Varje tentand rättar då en annan tentands tenta. Rättningen kommer att granskas av Lars.

- Om du lämnar flera svar på en uppgift tas det först givna som svar.

- För godkänd tentamen krävs godkänt på alla fyra avsnitt. Gränser och bonus anges i de enskilda avsnitten.

- En klarad KS ger dig godkänt på motvarande avsnitt.

- **Ni som gick kursen innan avsnittet om syntaxanalys tillkom behöver inte göra sista avsnittet på tentan.**

- Komplettering på ett avsnitt tillåts om man saknar en poäng för godkänt.

- **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

Alla skriver detta avsnitt och du behöver 10 poäng för att bli godkänd på det. Du får använda bonuspoäng från laborationerna C och Inet här.

1. Vad innebär *parametrisk polymorfism* i diskussioner om programmeringsspråk? (2p)

   (a) Kod är skriven för ej fullständigt bestämda typer.

   (b) Funktioner kan anta ett variabelt antal parametrar.

   (c) Samma funktionsnamn används i flera moduler.

   (d) Man tillåter funktioner att använda globala variabler.

2. Vad innebär *first class functions*? (2p)

   (a) Väldigt bra skriven kod.

   (b) Publika funktioner, d.v.s. en modul eller klass låter annan kod använda dem.

   (c) Funktioner betraktas som vilka värden som helst.

   (d) Kod med linjär tidskomplexitet.

3. Vilket påstående är korrekt om ett *typsäkert programmeringsspråk*? (2p)

   (a) Man får aldrig kompileringsfel.

   (b) Variabler kan inte anta otillåtna värden.

   (c) Typer härleds automatiskt, man behöver inte ange dem.

   (d) Typen följer värdet, man ska inte ange en variabels typ.

4. Vad innebär det att man programmerar med *strömmmar* i Unix? (2p)

   (a) Man styr om utdata till filer.

   (b) Man kopplar ihop olika programs ut-/indataströmmar (stdout/stdin).

   (c) Man anropar program som exekveras på andra datorer, även kallat ”remote procedure call”.

   (d) Man ger program olika prioritetsnivåer, vissa får exekvera i en snabbare ström än andra m.h.a. kommandot `nice`.

5. (a) Nämn två välkända *deklarativa* programmeringsspråk! (2p)

   (b) Språk som C, Java, m.fl. har *icke-strikt evaluering* i vissa väl valda fall. Förklara och ge exempel! (2p)

   (c) Klassificera följande språk som ”funktionellt”, ”imperativt”, eller ”objektorienterat”: Lisp, Fortran, C. (3p)

Funktionell programmering i Haskell

Du skriver detta avsnitt om du inte blev godkänd på KS i funktionell programmering. Det krävs 10 poäng för att bli godkänd på avsnittet. Du får använda bonuspoäng från F1 och F2 här.

6. Typen för uttrycket `map (\x -> x*x)` är... (2p)

   (a) `Num a => a`

   (b) `Num a => a -> a`

   (c) `Num a => [a] -> a`

   (d) `Num a => [a] -> [a]`

7. Vilket påstående är sant? (2p)

   (a) Haskell saknar skräpsamling (*garbage collection*).

   (b) Haskell saknar modulsystem.

   (c) Haskell kan inte evaluera strikt.

   (d) Haskell stödjer inte automatisk typkonvertering.

8. Förklara uttrycket `odd . length`. (2p)

9. Du ska skriva en funktion för "kryptering" av text på engelska med metoden ROT13. Den innebär att man

   - låter bokstäverna A till Z representeras av siffrorna 0 till 25,
   - för varje bokstav adderas talet 13 modulo 26,
   - innan man tolkar om det som en bokstav igen.
   - Andra tecken än bokstäver lämnas oförändrade.

   Som exempel ger denna metod att A blir ett N, B blir O, och Z blir M.

   (a) Skriv funktionen `rot13char :: Char -> Char` som utför ROT13 på precis en bokstav. Det räcker om bara stora bokstäver krypteras, men alla andra bokstäver ska lämnas oförändrade. (Se även ledningen nedan!) (2p)

   (b) Skriv funktionen `rot13 :: String -> String` som använder `rot13char` för att kryptera en sträng. (2p)

   Exempel:

```
ghc> rot13 "KTH"
"XGU"
ghc> rot13(rot13 "KTH")
"KTH"
ghc> rot13 "VAD BETYDER FRA?"
"INQ ORGLQRE SEN?"
```

Ledning: Till din hjälp har du funktioner från modulen Char

```
ord :: Char -> Int
chr :: Int -> Char
```

som konverterar till och från ASCII-värden. Bokstaven "A" har ASCII-värde 65.

Det är också bra att känna till

```
mod :: (Integral a) => a -> a -> a
```

som beräknar modulofunktionen (ex: $20 + 10 \bmod 26 = 4$).

10. Betrakta följande (något bristfälliga) definitioner för att lagra svenska och amerikanska adresser i samma system:

```
type Adressat = String
type Utdelningsadress = String
type Postkod = Int
type Postadress = String

type Town = String
type State = String
data Zip = Zip Integer
         | ZipPlus Integer Integer

data Adr = SvAdress Adressat Utdelningsadress Postkod Postadress
         | USAdress Adressat Utdelningsadress Town State Zip
```

(a) Skriv en funktion `svenskadress :: Adr -> Bool` som avgör om en adress i `Adr` är svensk eller inte. (1p)

(b) Skriv en funktion `stad :: Adr -> String` för att, givet en adress i datastrukturen `Adr`, returnera postadress (Sv) eller *town* (US). (2p)

(c) Skriv en funktion `svenskaorter :: [Adr] -> [String]` som returnerar de postadresser som kommer från svenska adresser. Du måste använda minst en högre ordningens funktion på ett meningsfullt sätt. (2p)

Exempel:

```
ghci> svenskadress (SvAdress  "Lasse" "Osquars Backe 2" 10044 "Stockholm")
True
ghci> svenskadress (USAdress "Larry" "OB 17" "Stockholm" "MN" (Zip 55321))
False
ghci> stad (SvAdress  "Lasse" "Osquars Backe 2" 10044 "Stockholm")
"Stockholm"
ghci> stad (USAdress "Larry" "OB 17" "Stockholm" "MN" (Zip 55321))
"Stockholm"
ghci> svenskaorter adresslista  -- Innehåller de två exemplen ovan
["Stockholm"]
```

Logikprogrammering med Prolog

Du skriver detta avsnitt om du inte blev godkänd på KS i logikprogrammering. Det krävs 10 poäng för att bli godkänd på avsnittet. Du får använda bonuspoäng från L1 och L2 här.

11. Vad kallas den form av logikprogrammering där man beskriver sitt problem i domänvariabler (*domain variables*), gärna över så kallade ändliga domäner, och samband mellan domänvariablerna, och använder en speciell lösningsfunktion för att leta upp en lösning till problemet? Du kan svara på engelska eller svenska. (2p)

12. Man använder *snitt* för att (2p)

    (a) undvika oönskad "backtracking".

    (b) hitta fler lösningar.

    (c) få svansrekursion i Prolog.

    (d) hantera I/O i Prolog.

13. Betrakta följande kod:

    ```
    predikatet(X, Z) :- pred1(X, Y).
    predikatet(X, Z) :- pred2(Y, Z).
    ```

    Vilket eller vilka påståenden är sanna? (2p)

    (a) Det räcker med att `pred1` *eller* `pred2` lyckas för att `predikatet` ska lyckas.

    (b) Det krävs att både `pred1` *och* `pred2` lyckas för att `predikatet` ska lyckas.

    (c) Precis *ett* av `pred1 och pred2` lyckas för att `predikatet` ska lyckas.

14. Skriv ett predikat `icketom` som lyckas om dess parameter är en lista med minst ett element. (2p)

    Exempel:

    ```
    | ?- icketom([]).
    no
    | ?- icketom([1, 2, 3]).
    yes
    ```

15. Betrakta koden nedan, där andra och tredje parametrarna ska betraktas som returvärden. Beskriv i ord vad predikatet `pred` beräknar. (2p)

    ```
    pred([X], X, X).
    pred([X|Xs], F, L) :- F = X, predhelp(Xs, L).

    predhelp([X], L) :- L = X.
    predhelp([X|Xs], L) :- predhelp(Xs, L).
    ```

16. Predikatet `summa` ska summera positiva tal i en lista, men det finns två fel i definitionen. Vilka? (2p)

```
summa([X|Xs], S) :-
        X =< 0, summa(Xs, S2).
summa([X|Xs], S) :-
        X > 0, summa(Xs, S2),
        S is S2 + X.
```

17. "Omvänd polsk notation" (OPN) är en postfixnotation för aritmetiska uttryck där man ger uttrycket som en lista av element, tal eller operatorer, som evalueras från vänster till höger. OPN har länge använts i de klassiska HP-räknarna och de grundläggande finessen är att man inte behöver oroa sig över precedensregler.

    När man evaluerar ett uttryck givet i OPN gör man så här:

    - Om nästa element är ett tal, lägg det på stacken.
    - Om nästa element är en operator (+, *, osv), applicera den på stackens två översta element och lägg dit resultatet istället.
    - Om listan är slut betraktas värdet på stacken som resultat.

    Du ska nu skriva predikatet `eva(L, Val)` som evaluerar ett OPN-uttryck givet i L och ger dig resultatet i Val. Du behöver bara implementera två operatorer, addition och multiplikation. Till din hjälp kan du använda predikat `number/1` som lyckas om dess argument är ett tal. (3p)

    Exempel:

```
| ?- number(17).
yes
| ?- number(+).
no
| ?- eva([17], Val).
Val = 17 ?
yes
| ?- eva([1,3,+], Val).
Val = 4 ?
yes
| ?- eva([1,2,+,3,4,+,*], Val).
Val = 21 ?
yes
```

    Ledning: Använd ett hjälppredikat med tre parametrar: L, Val, och S, där S är en lista som representerar din stack. Du behöver inte fundera på felhantering.

Syntaxanalys

Du skriver detta avsnitt om du inte blev godkänd på KS i syntaxanalys. Det krävs 10 poäng för att bli godkänd på avsnittet. Du får använda bonuspoäng från S1 och S2 här.

18. Beskriv i ord vad följande reguljära uttryck beskriver.

    (a) `a(aa)*` (1p)

    (b) `((ab)|(cd))*e` (1p)

    (c) `ATG((A|C|G|T)(A|C|G|T)(A|C|G|T))*((TAA)|(TAG)|(TGA))` (2p)

19. Ge reguljära uttryck för nedanstående språk. Du får *endast* använda operationerna konkatenering, alternering (med `|`), upprepning (med `*`), samt gruppering (med paranteser).

    (a) *Onda heltal*: de heltal i basen 10 som någonstans har följden "666". (1p)

    (b) *Binära tvåpotenser*: tal över $0, 1$ som representerar 2^k för $k \geq 0$. (1p)

20. Inom datalogin innebär *lexikal analys*... (2p)

    (a) att skriva reguljära uttryck.

    (b) att konvertera indata till en lista med symboler.

    (c) att avgöra om en text passar ihop med en grammatik.

    (d) att göra en stilistisk bedömning av ett program.

21. **Bakgrund:** I molekylärbiologin är molekyler kallade "mikro-RNA" (miRNA) ett mycket hett ämne. De har en speciell struktur som kan beskrivas som att en sträng av nukleotider (RNA:s byggstenar) A, C, G, och U sträcks ut och sedan böjs tillbaka, och då parar ihop byggstenar med varandra: A mot U och C mot G. I "bortre änden" går det inte att para ihop, så där får man en loopformation. Ibland får man också en liten utbuktning ("bulge") bland de ihopparade nukleotiderna. Se bilden nedan för ett trivialt exempel där GUAU...AUAU har vikts ihop över sig själv och de vertikala strecken markerar nukleotidpar.

```
(början)              U        U
     GUAUGUGCAUGCAUAUG CGCAUGUG  G
     |||||||||||||||| | ||||||
     UAUAUAUGUACGUGUAC-GUGUAUAC  U
(slut)                          A
```

    Uppgift: Vi kan skriva en grammatik för att beskriva miRNA-molekyler:

```
stem ::= 'A', innerstem, 'U'
       | 'G', innerstem, 'C'
       ;

innerstem ::= stem
```

```
      |  bulge
      |  loop
      ;

bulge ::= nucl, stem;

loop ::= nucl, loop
      |  nucl
      ;

nucl ::= 'A' | 'C' | 'G' | 'U';
```

Rotproduktionen är `stem`.

(a) Beskrivs följande strängar av grammatiken? Motivera dina tre beslut. (3p)

i. `AAACAAAUUUUUU`

ii. `AAAAAAUUUUUA`

iii. `ACCCCCCCCCCU`

(b) Den givna grammatiken tillåter hur långa loop-produktioner som helst. Skriv om grammatiken till att kräva att loopen använder fyra nukleotider. Förklara din modifikation. (2p)

(c) Är grammatiken prediktiv, dvs LL(1)? Ingen poäng utan motivering. (2p)