Tentamen i DD1361 Programmeringsparadigm, 29 november 2008

Kursansvarig: Lars Arvestad

- **Skriv inte namn** på tentaomslaget eller skrivpappren! Tentorna är anonyma! Istället skriver du sal och platsnummer. **Om du sitter på plats 17 i sal Q21 kan du märka dina papper med "Q21:17".**
- Detta är en **kamraträttad tenta**! Lämna in tentan när du är klar och lämna sedan salen. Återvänd till *din* plats kl 11.40 för gemensam rättning! Varje tentand rättar då en annan tentands tenta. Rättningen kommer att granskas av Lars.
- **Godkända hjälpmedel:** Skrivmedel. "Prolog programming" av Brna. "Programming in Haskell" av Hutton, eller det tidigare särtrycket av samma bok. "Functional Programming" av Jeroen Fokker.
- 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.

Lycka till!

Allmänna frågor

1. Ge två exempel på imperativa programmeringsspråk. (2p)
2. Ge två exempel på språk som inte använder *skräpsamling* (garbage collection). (2p)
3. Vad innebär *överlagring* (overloading) av funktioner? (2p)
4. Vad innebär *multipelt arv*? (2p)
5. Ge ett exempel på en *sidoeffekt* i ett imperativt språk. (2p)
6. I vissa språk kan man ge en funktion färre parametrar än den är definierad för, och få tillbaka en funktion som tar de kvarvarande parametrarna. Detta kallas ibland för partiell tillämpning (*partial application*), men i kursen har vi använt ett annat namn. Vilket? (2p)

Funktionell programmering i Haskell

7. Ange typsignatur för dessa Haskelluttryck:
   (a) `map length` (2p)
   (b) `[(x, odd x) | x <- [1..]]` (2p)

8. Definiera funktionen `and :: [Bool] -> Bool` med hjälp av rekursion. Funktionen returnerar `True` om alla elementen i en lista är sanna, och `False` annars. Den ska uppträda så här: (3p)

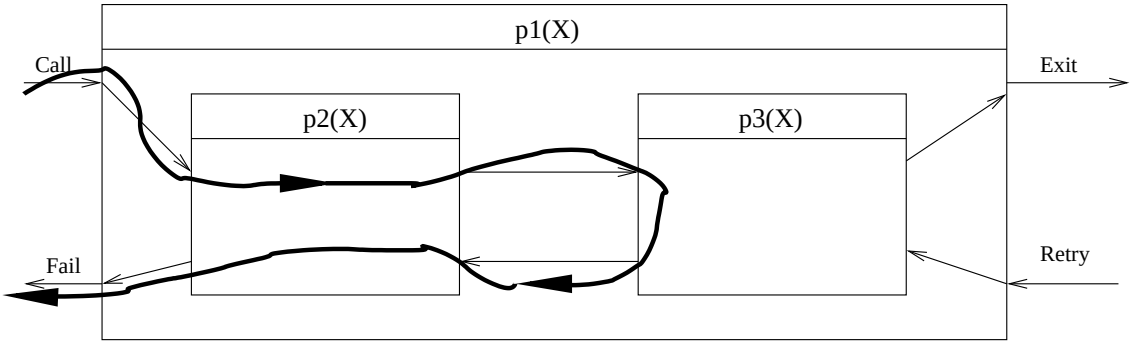
```
Hugs> and []
True
Hugs> and [True]
True
Hugs> and [True, True, True]
True
Hugs> and [True, False]
False
```

9. Skriv om nedanstående C-kod till en Haskell-funktion och ange dess typsignatur. Det är tänkt att `v1` och `v2` är vektorer i C, av längd `n`, och bör ersättas med listor i din Haskellkod. (4p)

```
int inre_produkt(int n, int *v1, int *v2)
{
   i = 0;
   sum = 0;
   while (i < n)
   {
       sum = sum + v1[i] * v2[i];
       i++;
   }
   return sum;
}
```

Logikprogrammering med Prolog

10. Figuren visar ett enkelt låddiagram och ett angivet flöde. Skriv Prolog-predikat, med fakta, som kan inducera detta diagram. (4p)



11. Förklara vad följande två Prologpredikat beräknar. (3p)

```
s(0, 0).
s(X, 1) :- X > 0.
s(X,-1) :- X < 0.

sv([], []).
sv([X|Xs], [S|Ss]) :- s(X,S), sv(Xs, Ss).
```

12. Skriv ett predikat `utan(K, L)` för att generera en lista med k icke-unifierade variabler. (2p)

    Exekveringsexempel:

```
| ?- utan(3, X).
X = [_A,_B,_C] ? n
no
| ?- utan(0, X).
X = [] ? n
no
```

Syntaxanalys

Du som följde kursen innan 2005 struntar i detta avsnitt!

13. Skriv ett regex som beskriver

    (a) språket {triathlon, quadrathlon, pentathlon}. (2p)

    (b) alla strängar över alfabetet A till Z, av udda längd. (2p)

14. Använd grammatiken här nedanför för att besvara dessa två uppgifter:

    (a) Skriv två meningar som använder orden ”jag”, ”sprang”, ”öh”, ”snabbt” och ”typ”, samt följer grammatiken. (2p)

    (b) Hur många olika meningar kan grammatiken producera om man får välja bland alla listade ord? (2p)

```
mening         ::= subjekt paus aktivitet
                 ;

aktivitet      ::= verb adverb generalisering
                 | generalisering paus verb adverb
                 ;

verb           ::= "sprang"
                 | "hoppade"
                 | "simmade"
                 ;

adverb         ::= "snabbt"
                 | "långsamt"

subjekt        ::= "jag"
                 | "han"
                 | "hon"
                 ;

paus           ::= "öh"
                 | "eh"
                 | "bah"
                 ;

generalisering ::= "typ"
                 | "liksom"
                 ;
```