KTH Information and
Communication Technology

ID1019
Johan Montelius

Programmering II (ID1019) utkast

Namn: _____

Instruktioner

- Svaren skall lämnas på dessa sidor, använd det utrymme som finns under varje uppgift för att skriva ner ditt svar.
- Svar skall skrivas på svenska eller engelska.
- Du skall lämna in hela denna tentamen.
- Inga ytterligare sidor skall lämnas in.

Betyg

Tentamen består av två delar. En grundläggande del om fem frågor (1-5) och en del för högre betyg på fyra frågor (6-9). De första fem frågorna handlar om grundläggand funktionell programmering: mönstermatchning, rekursion, icke-modifierbara datastrukturer mm. Dessa fem frågor skall besvaras tillfredsställande (8 av 10 poäng) för betyg E (7 poäng för FX).

De övriga fyra frågorna (6-9) handlar om: semantik, högre-ordningens funktioner, komplexitet, processer mm. Betygsgränserna för de högre betygen baseras enbart på dessa frågor men ges endast (och rättas enbart) om den grundläggande delen har besvarats tillfredsställande

- D: 1 poäng
- C: 2 poäng
- B: 3 poäng
- A: 4 poäng

Namn:_____ Persnr:_____

1 dubbla jämna [2p]

Implementera en funktion som tar en lista av tal och returnerar en liknande lista men där alla jämna tal har dubblerats. Du kan använda dig av funktionen `rem(n,k)` som returnerar resten när n divideras med k.

2 ett binärt träd [2p]

Implementear en funktion, `sum/1`, som tar ett träd och returnerar summan av alla värden i trädet. Trädet är representerat som följer:

```
@type tree :: {:node, integer(), tree(), tree()} |  nil
```

Namn:_____ Persnr:_____

3 spegla ett träd [2p]

Implementera en funktion, `mirror/1`, som spegelvänder ett binärt träd. Ett träd spegelvänds genom att spegelvända, och byta plats på, dess två grenar.

4 add/2 [2p]

Implementera funktionen `add/2` som adderar ett tal till en heap. En heap är ett träd med sitt största element i dess rot där dess två grenar består av heap:ar.

För att hålla heapen balanserad så skall man växla höger och vänster grenar dvs, när vi skall lägga till ett element i en undergren så lägger vi till det i högra grenen men gör resultatet till den nya heapens vänstra gren.

- `@spec add(heap(), integer()) :: heap()`

Namn:_____ Persnr:_____

5 Fizz-Buzz [2p]

Vi skall implementera en funktion `fizzbuzz/1` som givet ett tal $n \geq 0$ returnerar en lista av de n första elementen i fizzbuzz-serien. Fizz-Buzz går till så att man räknar från 1 till n men byter ut all tal som är delbara med 3 mot `:fizz`, alla tal som är delbara med 5 med `:buzz` och all tal som är delbara med både 3 och 5 mot `:fizzbuzz`. De första fem elementen är således: `[1,2,:fizz,4,:buzz]`.

Du skall implementera den rekursiva funktionen `fizzbuzz/4` som hjälper oss att göra detta. Första argumentet är nästa element i listan, andra för att veta att vi är klara och tredje o fjärde håller reda på om talet är delbart med 3 eller 5. Du får bara använda addition, inte någon annan aritmetisk operation. Du skall inte göra funktionen svansrekursiv.

Om du tycker att detta är enkelt så prova att göra det i kväll efter tre öl!

```
def fizzbuzz(n) do fizzbuzz(1, n+1, 1, 1) end
```

Namn:_____ Persnr:_____

6 Formell semantik [1p]

Lambdakalkyl

Evaluera följande lambdauttryck:

- $(\lambda x \rightarrow x + 5)4$
- $(\lambda x \rightarrow (\lambda y \rightarrow x + 2 * y)3)5$
- $(\lambda x \rightarrow (x)5)(\lambda z \rightarrow z + z)$

Operationell semantik

Någonting om operationell semantik.

Namn:_____ Persnr:_____

7 reducera ett träd [1p]

reduce/3

Implementera en funktion `reduce(tree, init, op)` som tar ett binärt träd, med värden i noderna, ett initialt värde och en operation, `op`, som tar två argument; resultatet av en reducering och värdet i en nod. Funktionen `reduce/3` skall returnera det reducerade värdet som fås då man tillämpar operationen rekursivt:

- det reducerade värdet av ett tomt träd är det accumulerade värdet
- det reducerade värdet av en nod är: värdet av den reducerade högergrenen - där det accumulerade värdet är operatorn applicerad på det reducerade värdet av vänstergrenen och nodens värde.

Trädet är representerat som följer:

```
@type tree :: {:node, any(), tree(), tree()} |  nil
```

to_list/1

Implementera en funktion `to_list/1`, som tar ett träd och returnerar en lista av alla element i trädet i infix-ordning. Funktionen skall implementeras med hjälp av `reduce/3`.

Namn:_____ Persnr:_____

8 en kö [1p]

Antag att vi representerar en kö med hjälp av två listor och har nedanstående implementering av `enqueue/2` och `dequeue/1`. Vad är den amorterade tidskomplexiteten för att lägga till och sedan ta bort ett element ur kön?

```
def enqueue({:queue, head, tail}, elem) do
  {:queue, head, [elem|tail]}
end

def dequeue({:queue, [], []}) do :fail end
def dequeue({:queue, [elem|head], tail}) do
  {:ok, elem, {:queue, head, tail}
end
def dequeue({:queue, [], tail}) do
  dequeue({:queue, reverse(tail), []})
end
```

Namn:_____ Persnr:_____

9 parallel summering [1p]

Implementera en funktion `sum/1` som tar ett binärt träd med tal i löven och summerar alla talen i trädet parallellt.