KTH Information and
Communication Technology

ID1019
Johan Montelius

Programmering II (ID1019)
2019-06-05

Instruktioner

- Svaren skall lämnas på dessa sidor, använd det utrymme som finns under varje uppgift för att skriva ner ditt svar (inte på baksidan).
- 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. De första fem frågorna handlar om grundläggande funktionell programmering: mönstermatchning, rekursion, icke-modifierbara datastrukturer mm. Den första delen är grundkravet för att klara kursen:

- FX: 7 poäng
- E: 8 poäng

Den andra delen, 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 (8 av 10 poäng):

- D: en fråga korrekt besvarad
- C: två frågor korrekt besvarade
- B: tre frågor korrekt besvarade
- A: alla frågor korrekt besvarade

Namn:_____ Persnr:_____

1 plocka bort var n:te [2p]

Implementera en funktion, `drop/2` som tar en lista och ett tal $n > 0$, och returnerar en lista där var n:te element har plockats bort.

Exempel: `drop([:a,:b,:c,:d,:e,:f,:g,:h,:i,:j], 3)` skall ge svaret `[:a,:b,:d,:e,:g,:h,:j]`.

2 rotera en lista [2p]

Implementera en funktion `rotate/2` som tar en lista, av längd l, och ett tal n, där $0 \leq n \leq l$, och returnerar en lista där elementen roterat n steg.

Du kan använda dig av två biblioteksfunktioner: `append/2` och `reverse/1` (inte `++`). Din lösning får dock bara använda dessa funktioner en gång vardera under en evaluering.

Exempel: `rotate([:a,:b,:c,:d,:e], 2)` returnerar `[:c,:d,:e,:a,:b]`.

Namn:_____ Persnr:_____

3 n:te lövet [2p]

Implementera en funktion `nth/2` som hittar värdet i det n'te lövet i ett binärt träd traverserat djupet först vänster till höger. Funktionen skall ta ett tal $n > 0$ och ett binärt träd och returnera antingen:

- `{:found, val}` om det n'te lövet hittas och har värdet `val` eller,
- `{:cont, k}` om endast $n - k$ löv hittades dvs man skulle behöva k fler löv för att hitta det n'te.

Du skall inte tranformera hela trädet till en lista och sedan hitta det n'th lövet. Du skall hitta lövet genom att traversera trädet och stanna så snart du hittar det n'te lövet.

Träden är representerade enligt följande, notera att det inte finns något tomt träd och att vi endast har värden i löven:

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

Exempel:

```
nth(3, {:node, {:node, {:leaf, :a}
                       {:leaf, :b}}
               {:leaf, :c}})
```

skall returnera `{:found, :c}`

Namn:_____ Persnr:_____

4 HP35 och omvänd polsk notation [2p]

Världens bästa kalcylator är naturligtvis HP35 som använder omvänd polsk notation. Man skriver inte `2 + 3 =` utan `2 3 +` och får genast svaret 5. Varje gång man skrev in ett tal så las det på stacken. Tryckte man på en binär operator så ersattes de två översta värdena på stacken med resulatet av operationen. Knappade man in `3 4 + 2 -` fick man svaret 5 och `3 4 + 2 1 + -` resulterade naturligtvis i 4.

Implementera en funktion `hp35/1` som tar en sekvens av instruktioner och returnerar resultatet. Instruktioneran består av tal eller operatorer och kan naturligvis vara av godtycklig längd. Du behöver inte hantera sekvenser som inte är giltiga utan vi förutsätter att sekvensen representerar ett aritmetiskt uttryck.

```
@type op() :: :add | :sub
@type instr() :: integer() | op()
@type seq() :: [instr()]

@spec hp35(seq()) :: integer()
```

Exempel: anropet `hp35([3, 4, :add, 2, :sub])` skall returnera `5` eftersom $(3 + 4) - 2 = 5$.

Namn:_____ Persnr:_____

5 Pascals triangel [2p]

Implementera en funktion, `pascal/1` som tar ett tal n (> 0) och returnerar den n'te raden i Pascals triangel. Nedan ser du hur triangel är uppbyggd: den första raden är alltså `[1]` och den femte raden `[1, 4, 6, 4, 1]`. Funktionen skall naturligtvis kunna generera vilken rad som helst, inte bara de som visas.

Tänk rekursivt.

```
              [1]
            [1 , 1]
          [1,  2,  1]
        [1,  3 , 3,  1]
      [1,  4,  6,  4,  1]
    [1, ...              1]
```

Exempel: `pascal(5)` returnerar `[1,4,6,4,1]`.

Namn:_____ Persnr:_____

6 Formell semantik [P/F]

evalueringsordning

Elixir är ett funktionellt språk där argumenten till ett funktionsanrop evalueras innan vi evaluerar funktionen. Detta är vad vi kallar *ivrig evaluering* (*eager evaluation*). Andra funktionella språk använder så kallad *lat evaluering* och evaluerar argumenten endast om de verkligen behövs.

Om vi håller oss till Elexirs funktionella del (utan processer eller sidoeffekter) så är det kanske så att det saknar betydelse då svaret vi får blir det samma.

Vi kan ta definitionen av `test/2` nedan som exemepl; anropet `test(3 + 4, fib(17))` ger alltid svaret `7` oavsett vilken orning saker görs på. Är det verkligen i detta fall kvitt samma om vi använder ivrig eller lata evaluering? Motivera ditt svar.

Ge ett exempel då resultatet (eller brist på) verkligen blir annorlund beroende på om vi använder ivrig eller lat evaluering. Motiver ditt svar.

```
def test(one, two) do
   if  one > 10 do
      two
   else
      one
   end
 end
```

Namn:_____ Persnr:_____

7 Transformera ett träd [P/F]

Implementera en funktion `trans/2` som tar ett träd och en funktion som argument och returnerar ett nytt träd som är isomorft med det givna trädet men där nodernas värden har transformerats med hjälp av den givna funktionen.

Implementera sedan en funktion, `remit/2` som tar ett träd och ett tal n som argument och returnerar, genom att använda sig av `trans/2`, ett träd där varje värde x har bytts mot resten vid heltalsdivision med n (fås genom anropet till `rem(x,n)`).

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

@spec trans(tree(), (any()  -> any()))
```

Namn:_____ Persnr:_____

8 en enkelbiljett [P/F]

Det är ju populärt att åka tåg i Europa i år så varför inte skriva ett program som hittar kortaste avståndet mellan två städer. Vi antar att vi har en karta som beskriver alla linjer mellan olika städer i Europa. Givet en stad kan vi med hjälp av kartan plocka ut en lista över dess närmaste grannar och avstånden till dessa.

Skelletet nedan (och på andra sidan) är ett program som fungerar även om kartan inehåller cirkulära strukturer. Det är också hyggligt effektivt eftersom det använder sig av dynamisk programmering. Skriv klart programmet.

```
@type city() :: atom()
@type dist() :: integer() | :inf

@spec shortest(city(), city(), map()) :: dist()

def shortest(from, to, map) do

      ....           =  Map.new([{to, 0}])

      ....           =  check (from, to, ....       ,....        )

  dist
end

@spec check(city(), city(), map(), map()) ::  {:fond, dist(), map()}

def check(from, to, ....       , ....         ) do

  case      ....                    do

    nil ->

      shortest(from, to, ....        , ....         )

    distance ->

      ....

  end
end
```

Namn:_____ Persnr:_____

```
@spec shortest(city(), city(), map(), map()) :: {:found, dist(), map()}

def shortest(from, to, ....     , ....        ) do

  ....                 = Map.put(....     , ....      , :inf)

  ....                 = Map.get(....     , from)

  ....                 =  select(neighbours, to, updated, map)

  ....

  {:found, dist, updated}
end

@spec select([{:city, city(), integer()}], city(), map(), map()) ::
         {:found, dist(), map()}

def select([], _,  ....    , ....      ) do  ....                end

def select([{:city, next, d1} | rest], to,   .....  , .....       )    do

        ....        = check(next, to,  ....     , ....          )

  dist = add(d1,d2)

        ....        = select(rest, to,  ....    , ....           )

  if sele  < dist do
    {:found, sele, updated}
  else
    {:found, dist, updated}
  end
end

@spec add(dist(), dist()) :: dist()

def add(  ....  , ....       ) do  ....        end
def add(  ....  , ....       ) do  ....        end
def add(  ....  , ....       ) do  ....        end
```

Namn:_____ Persnr:_____

9 HP35 på återseende [P/F]

Eftersom HP35 är en så förträfflig miniräknare skall du implementera en process som beter sig som en sådan. Processen skall ta emot meddelanden som består av en tal eller operationer. Du behöver bra implementera addition men vi will ha tillbaks resultatet när vi utför en addition.

Implementera processen och ge en testfunktion som visar hur man kan använda räknaren för att addera två tal.

Svara på nästa sida.

Namn:_____ Persnr:_____

Svar uppgift 9.

Namn:_____ Persnr:_____

Appendix

Map

- `put(map, key, value)` put(map(), key(), value()) :: map() Puts the given value under key in map.
- `get(map, key, default \\nil)` get(map(), key(), value()) :: value() Gets the value for a specific key in map. If key is present in map with value value, then value is returned. Otherwise, default is returned (which is `nil` unless specified otherwise).
- `new(enumerable)` new(Enumerable.t()) :: map() Creates a map from an enumerable. Duplicated keys are removed; the latest one prevails.