KTH Information and
Communication Technology

ID1019
Johan Montelius

Programming II (ID1019) 2016-06-10 09:00-13:00

7.5 credits

Name: _____

Instruction

- You are not allowed to have any material besides pen and paper. Mobiles etc, should be left to the guards.
- All answers should be written in these pages, use the space allocated after each question to write down your answer.
- Answers should be written in Swedish or English.
- You should hand in the whole exam.
- No additional pages should be handed in.

Grades

The exam is divided into a number of questions where some are a bit harder than others. The harder questions are marked with a star *points**, and will give you points for the higher grades. The exam is thus divided into basic points and points for higher grades. First of all make sure that you pass the basic points before engaging with the higher points.

Note that, of the 40 basic points only at most 34 are counted, the points for higher grades will not make up for lack of basic points. The limits for the grads are as follows:

- Fx: 22 basic points
- E: 24 basic points

- D: 30 basic points
- C: 34 basic points
- B: 34 basic points and 14 higher points
- A: 34 basic points and 20 higher points

The limits could be adjusted to lower values but not raised.

Gained points

Don't write anything here.

Question	1	2	3	4	5	6	Σ
Max B/H	4/-	10/2	2/6	4/2	4/4	16/10	40/24
B/H							

Total number of points:

Grade:

Name:_____

1 Data structures and pattern matching

1.1 what is Y [2 points]

What is the resulting binding for `Y` i the following pattern matching expressions (each one by its own), in the case that the matching succeeds:

- `[Y|_] = [1,2,3]` **Answer:** `Y = 1`
- `[X,_ |Y] = [1,2,3]` **Answer:** `Y = [3]`
- `Y = [1|Y]` **Answer:** *misslyckas*
- `[X,Y,Z] = [1|[2,3]]` **Answer:** `Y = 2`
- `Z = 32, X = {foo, Z}, {Y, _} = X` **Answer:** `Y = foo`

1.2 Constructor, selector and a paint job [2 points]

Assume that we represent cars using a tuple `{car, Is, Brand, Color}`, where `Id` is a unique identifier, `Brand` a string denoting the brand and `Color` a color represented by an atom for example `green`, `black` etc.

Implement a function `new/3` that takes an identifier, a brand and a color and returns a car.

Answer:

```
new(Id, Brand, Color) -> {car, Id, Brand, Color}.
```

Implement a function `color/1` that takes a car and returns the color of the car.

Answer:

```
color({car, _, _, Color}) -> Color.
```

Implement a function `paint/2` that takes a car and a color and returns an identical car but in the new color.

Answer:

```
paint({car, Id, Brand, _}, Color) -> {car, Id, Brand, Color}.
```

Name:_____

2 Recursive functions

2.1 Sum and length on one go [2 points]

Assume we have a list of integers and we want to calculate the sum of all integers and the length of the list but we want to do it in one pass.

Implement a function `once/1` that takes a list and returns a tuple `{Sum, Length}` with the sum of the integers and the length of the list.

You're not allowed to give a solution that first runs through the list to calculate the sum and then runs through the list a second time to calculate the length.

Answer:

```
once([]) -> {0,0};
once([H|T]) ->
   {S,L} = once(T),
   {S+H,L+1}.
```

2.2 The Ackermann function [2 points]

All functions that we have worked with during the course have been so called *primitive recursive functions*. There are however recursive functions that are not primitive and grow faster than the exponential functions we warned about. The most well known function that is not primitive recursive is the *Ackermann function* and is defined as follows:

$$Ack(m,n)\begin{cases} n+1 & \text{if } m = 0 \\ Ack(m-1,1) & \text{if } m > 0 \text{ and } n = 0 \\ Ack(m-1, Ack(m,n-1)) & \text{if } m > 0 \text{ and } n > 0 \end{cases}$$

Implement a function `ack/2` given the definition of the Ackermann function.

Answer:

```
ack(0,N) -> N+1;
ack(M,0) -> ack(M-1, 1);
ack(M,N) -> ack(M-1, ack(M, N-1)).
```

Ack(4,1) is equal to 65533 and takes aprx 30 seconds to compute on my laptop. Ack(4,2) is .... and takes .....

Name:_____

- Yes,how long time would it take? An extra bonus point to those who are even in the vicinity of the either the answer or the runtime.

Note - do not spend the rest of the exam trying to calculate the answer!

Answer: Ack(4,2) är i storleksordningn $2^{2^{16}}$, ett tal som har cirka 20.000 siffror om vi skriver ut det decimalt. Det skulle ta i storleksordningen $4^{2^{16}}$ sekunder att räkna ut. Det är i närheten av $2^{2^{15}}$ år eller, universum är 14 miljarder år, skulle vara $2^{2^{10}}$ gånger universums ålder.... har jag räknat rätt nu?Ack(4,2) is in the order of $2^{2^{16}}$, a number that has 20.000 digits if we write it in decimal notation. It would take aprx $4^{2^{16}}$ seconds to calculate. This is close to $2^{2^{15}}$ years or, the age of the universe is 14 billion years, is $2^{2^{10}}$ times the age of the universe..... did I get it right?

2.3 an arithmetic expression [2 points]

Assume that we choose to represent an arithmetic expression as a list of integers and operators. To avoid ambiguities we limit the operations to '+' and '-'. A representation of teh expression $5+2-3+10$ is thus represented by the list:

```
[5, '+', 2, '-', 3, '+', 10]
```

Implement a function `eval/1` that takes an arithmetic expression and returns the calculated result. The example above should return `14`. We assume that the expression is correct.

Answer:

We do not want to evaluate $5+2-3+10$ as $5+(2-(3+10))=-6$ but as $((5+2)-3)+10=14$.

```
eval([]) -> 0;
eval([N|T]) -> eval(T, N).

eval([], Sum) -> Sum;
eval(['+', N2 | L], Sum) -> eval(L, Sum + N);
eval(['-', N2 | L], Sum) -> eval(L, Sum - N).
```

2.4 isomorfic trees 2 points]

Assume we represent the empty tree by `nil` and nodes in the tree by tuples `{tree, Value, Left, Right}` where `Left` and `Right` are trees.

Name:_____

Two trees are *isomorfic* if the are both empty or if they are both nodes where their right brances are isomorfic and their left branches are isomorfic. The value of the nodes are not important, it is the structure that we are looking for.

Name:_____

Implement a function `isomorfic/2` that returns `true` or `false` depending on if its two arguments are isomorfic or not.

Answer:

```
isomorfic(nil, nil) -> true;
isomorfic({tree, _, L1, R1}, {tree, _, L2, R2}) ->
   case isomorfic(L1, L2) of
       true -> isomorfic(R1, R2);
       false -> false
   end;
isomorfic(_, _) -> false.
```

2.5 mirror image[2 points]

Assume that we represent a tree as descibed below. A mirror image of a tree is of course a tree where each node has a left branch that is the mirror image of the original right branch and vice verse.

Implement a function `mirror/1` that takes a tree and returns the mirror image of the tree.

```
-type tree() :: nil | {tree, any(), tree(), tree()}.
```

Answer:

```
mirror(nil) -> nil;
mirror({tree, Value, Left, Right}) ->
   {tree, Value, mirror(Right), mirror(Left)}.
```

2.6 calculation of polynomial [2 points*]

Assume that we represent a polynomial as a list of the coefficients. The polynomial $2x^3 + 5x^2 + 7$ would be represented by the list `[2,5,0,7]`; the third coefficients is 0 since there is no term ax.

Implement a function `calc/2` that takes a polynomial and the value of the variable of the polynomial, and calculates the value of the polynomial. Note

that the polynomial can be of arbitrary degree, even of degree 0 when it only consist of a list of one value, the value of the constant. We can assume that there is at least one coefficient, the empty list is not a polynomial.

Answer:

```
calc(Poly, X) ->  calcp(Poly, X, 0).

calcp([], _, Sum) ->  Sum;
calcp([K|Poly], X, Sum) ->
    calcp(Poly, X, Sum*X + K).
```

3 Evaluating expressions

We have during the course worked with describing how a language can be defined by formally describing which terms, expressions and data structures we have and how we by rules can describe what should happen when we evaluate expressions. The following questions assume that we have defined a small language given the guidelines we have presented.

3.1 free variables [2 points]

Which is/are the free variable/variables in the following sequence?

```
Z = 42, Y, F = fun(X) -> foo(X, Y) end, F(Z)
```

Answer: `Y`

What is the result of evaluating the following sequence?

```
{X,Y} = {a, b}, F = fun(X) -> {X, Y} end, F(c)
```

Answer: `{c,b}`

Name:_____

3.2 pattern matching [2 points*]

The three first rules for pattern matching are:

- $P\sigma(a, s) \rightarrow \sigma$ if $a \equiv s$
- $P\sigma(\_, s) \rightarrow \sigma$
- $P\sigma(v, s) \rightarrow \sigma$ if $v/s \in \sigma$

What should we do with $P\sigma(v, s)$ when $v/t \notin \sigma$? Write a rule that describes how the pattern matching should be evaluted.

Answer: $P\sigma(v, s) \rightarrow \{v/s\} \cup \sigma$ if $v/t \notin \sigma$

3.3 λ expressons [4 points*]

Assume that we want to extended the language with lambda expressions. These are given a syntax on the form:

⟨*lambda*⟩ ::= 'fun' '(' ⟨*param*⟩ ')' '->' ⟨*sequence*⟩ 'end'

The *parameters*, `<param>`, is a sequence of variables and the *body*, `<sequence>`, is a sequence of expressions.

A lambda expression is evaluated to a so called *closure* that consists of its *parameters*, *body* and an environment θ. What does this environment θ consist of?

Answer:

4 Complexity

In the following questions it is important that you describe what for example n is and that you motivate your answer.

4.1 mirror image of a tree [2 points]

In you mirror a tree. What is the asymptotic time complexity for the function?

Answer:

4.2 complexity of `foo/2` [2 points]

What is the asymptotic time complexity for the function `foo/2`? Motivate your assumptions and alternative answers.

Name:_____

```
foo(_, nil) -> false;
foo(K, {node, K, V, L, R}) -> {ok, V};
foo(K, {node, E, _, L, R}) when K < E -> foo(K, L);
foo(K, {node, E, _, L, R}) -> foo(K, R).
```

Answer:

4.3 find a path [2 points*]

Assume we have a directed asyclic graph that is represented by tuples `{node, Id, Links}` where `Link` is a list of nodes that the node has connections to. We are somewhat naive and implement a search function as follows:

```
search(_, []) -> fail;
search(To, [{node, To, _}|_]) -> {ok, [To]};
search(To, [{node, From, Links}| Alt]) ->
   case search(To, Links) of
      {ok, Path} -> {ok, [From|Path]};
      fail -> search(To, Alt)
   end.
```

The function will not find the shortest path but it will find a path if there exists one. What is the asymptotic time complexity for the function?

Answer: Den asymptotiska tidskomplexiteten är $O(2^n)$ där n är antalet noder i grafen. Detta eftersom den graf vi ser i varje rekursionssteg visserligen är strikt mindre men proportionell till $n-k$ där k är rekursionsdjupet. Om vi har en förgreningsfaktor på f så har vi ett arbete i varje nod som är f gånger arbetet i underliggande noder. Vi får då ett totalt arbete proportionellt till f^n eller $O(2^n)$.

5 Concurrency

5.1 [2 points]

Implement a procedure `collect/0` that accepts a sequence of messages that are terminated by the message `done`. The procedure should return all the messages (except `done`) as a list where the messages are ordered in the order that they were received (firts message that was received first in the list.

```
-spec collect() -> [any()].
```

Answer:

Name:_____

```
collect() ->
  receive
     done -> [];
     X - > [X| collect()]
  end.
```

5.2 tic-tac-toe [2 points]

Assume that we have the following definition of `first/1`, `second/1` and `last/1`.

```
first(P) ->
  receive
     tic -> P ! tic, second(P)
     tac -> P ! tac, second(P)
  end.

second(P) ->
  receive
    tic -> P ! tic, last(P)
    tac -> P ! tac, last(P);
    toe -> P ! toe, last(P)
  end.

last(P) ->
  receive
     X -> P ! X, P ! done
  end.
```

Also assume that we have implemented the procedure `collect/0` in the previous question. What is the result when we evaluate the call `test()`?

```
test() ->
   Self = self()
   P = spawn(fun()-> first(Self) end),
   P ! toe,
   P ! tac,
   P ! tic,
   collect().
```

Answer: `[tac, toe, tic]\`

Name:_____

5.3 a monitor [4 points*]

A so called *monitor* is a construction that protects a critical section in a program. Executing threads that want to enter the critical section need to do this through the API offered by the monitor. Only one thread at a time is allowed to enter the critical section and threads that request entry should be granted access in the order that they requested entry.

How would you impelement a *monitor* in Erlang? Give an example, with code, that explains how a monitor could be implemented.

Answer:

```
new() -> spawn(fun() -> monitor() end.

monitor() ->
   receive
     {do, This} ->
         critical(This),
         monitor()
     end.
```

Name:_____

6 Programming

6.1 Huffman coding

6.1.1 decode [6 points]

Assume that we have a tree to decode Huffman coded text, where the coded text is represented as a sequence of zeros and ones. The nodes in the tree are represented by a tuple `{huf, Zero, One}` and the leaves by the tuple `{char, Char}`. When you decode a coded text you descend the tree and choose the left or right branch depending on if you read a zero or one. If you reach a leaf you have found the decoded character and can continue from the root of the tree.

Implement a function `decode/2` that takes a coded text and a decoding tree, and returns the decoded text.

Answer:

```
decode(Seq, Tree) -> decode(Seq, Tree, Tree).

decode([], _, _) -> [];
decode([0|T], {huf, Zero, _}, Tree) -> decode(T, Zero, Tree);
decode([1|T], {huf, _, One}, Tree) -> decode(T, One, Tree);
decode(Seq, {char, Char}, Tree) -> [Char|decode(Seq, Tree, Tree)].
```

Name:_____

6.1.2 build the tree [2 points*]

Assume that we have a frequency table represented as a list of tuples `{freq, {char, Char}, F}` and want to build a Huffman tree on the form given in the previous question. The table is ordered with the characters of the lowest frequency first. Your task is to implement a function `huffman/1` that takes the frequency table and returns a Huffman tree. To help you, you have a function `insert/2` that takes an element, `{freq, any(), integer()}` and a sorted table and returns an updated table where the element has been inserted at the right position.

Answer:

```
huffman([{freq, Tree, _}]) -> Tree;
huffman([{freq, A, Fa}, {freq, B, Fb} | Rest]) ->
       huffman(insert({freq, {huf, A, B}, Fa+Fb}, Rest)).
```

Name:_____

6.2 Mandelbrot

6.2.1 a computing process [6 points]

Assume that we have a function `mandel/2` that can calculate the "depth" of a point in the complex plane. The function takes a complex number and a maximal depth and returns either `0` or the depth where we can determine if the point does not belong to the Mandelbrot set. We also have a function `color/1` that takes a depth and returns a color.

Implement a process that when it is started attaches to a coordinator and helps in calculating the colors of the points that are sent by the server. The process should be started by the procedure `start/1` that takes the process identifier of the coordinator as an argument. It should send the message `{request, Pid}` to the coordinator and be prepared to help in the calculation.

The following two messages should be handled:

- `{calc, Pixle, Point, Max, Collector}`: the process should return `{color, Pixle, Color}` to the process `Collector`. Here `Pixle` is a representation of the pixel that is being calculated and `Point` is the corresponding point in the complex plane.
- `done` : the process should terminate.

Answer: Depending onm how the question is interpreted:

```
start(Cord) ->
  spawn(fun() -> init(Cord) end).

init(Cord) ->
  client(Cord).

client(Cord) ->
   Cord ! {request, self()}
   receive
      {calc, Pixle, Point, Max, Collector} ->
          Collector ! {color, Pixle, color(mandel(Point, Max))},
          client(Cord);
      done -> ok
   end.
```

Name:_____

6.2.2 a coordinator [2 points*]

Implement a coordinator that calculates a Mandelbrot image with the help of one or more computing processes. The coordinator should, when it is stated: start a process that should collect all computations and one procecess that should distribute work. When the collector has collected the whole image it should be sent to a process that is given when the coordinator starts.

Assume that the collector `collect/2`, and the distributor `server/7` are given. They have the following properties:

- `collect(Width, Height)` : waiting for colors from compyting processes and returns the complete image.
- `server(Width, Height, X, Y, K, Max, Collector)` : distributes work to computing processes to calculate an image with the dimension `Width`, `Height`, left upper corner at position `X`, `Y`, step length `K` and maximal depth `Max`. The computing processes should send their results to the `Collector`.

How do you implement the up start procedure? The procedure `start/7` should take the follwing arguments:

- `Width` och `Height` : the width and height of the image
- `X`, `Y` och `K`: position of teh upper left corner
- `Max` : maximum depth of the calculation.
- `Ctrl` : the process that should receive the final picture.

Answer:

```
start(Width, Height, X, Y, K, Depth, Cntr) ->
   spawn(fun() -> init(Width, Height, X, Y, K, Depth) end).

init(Width, Height, X, Y, K, Depth) ->
    Collector = spawn(fun() -> init_collector(Width, Height, Ctrl) end),
    server(Width, Height, X, Y, K, Depth, Collector).

init_collector(Width, Height, Ctrl) ->
   Image = collect(Width, Height),
   Ctrl ! {image, Image}.
```

Name:_____

6.3 a [total 4 + 6 points]

In this task you should implement a server that keeps track of a sum. It is reached over HTTP and will add numbers to an internal counter so that we can ask what the sum is.

We start with a simple server that will answer on port 8080. The procedure `request/1` will take care of requests and return an answer to the client. The idea is that we should be able to send two types of messages {add, N}, where `N` is an integer, and `sum` that is a request to know the total sum. We use HTTP to code the questions and answers but this is nothing that you have to consider to solve the task.

Note - the code that follows is not a solution.

```
-define(Port, 8080).

start() ->
    spawn(fun() -> init() end).

init() ->
    case gen_tcp:listen(?Port, [list, {active, false}, {reuseaddr, true}]) of
        {ok, Listen} ->
            handler(Listen),
            gen_tcp:close(Listen);
        {error, Error} ->
            io:format("oh no: ~w~n", [Error])
    end.

handler(Listen) ->
    case gen_tcp:accept(Listen) of
        {ok, Client} ->
            request(Client),
            handler(Listen);
        {error, _} ->
            error
     end.
```

Name:_____

6.3.1 a simple solution [4 points]

The given code is not a solution since it will simply echo the request that is received. You first task is to rewrite the server so that it does keep track of how much has been added and will return the sum to anyone that sends in a request. You don't have to re-write the server, simply add the changes that you need to the code given.

```
request(Client) ->
    case gen_tcp:recv(Client, 0) of
        {ok, Str} ->
            Response = case parse(Str) of
                           {add, N} ->
                               reply("received " ++ integer_to_list(N));
                           sum ->
                               reply("you want to know the sum");
                           error ->
                               notfound()
                       end,
            gen_tcp:send(Client, Response);
        {error, Error} ->
            error
    end,
    gen_tcp:close(Client).
```

Answer:

```
               :
            handler(Listen, 0),
               :


handler(Listen, N) ->
    case gen_tcp:accept(Listen) of
        {ok, Client} ->
            N1 = request(Client, N),
            handler(Listen, N1);
        {error, _} ->
            error
     end.

request(Client, Sum) ->
    {_, Updated} = case gen_tcp:recv(Client, 0) of
        {ok, Str} ->
```

Name:_____

```
            {Response, Added} = case parse(Str) of
                                    {add, N} ->
                                        {reply("received " ++ integer_to_list(N)), Sum+N};
                                    sum ->
                                        {reply("the sum is " ++ integer_to_list(Sum)), Sum});
                                    error ->
                                        {notfound(), Sum}
                                end,
            gen_tcp:send(Client, Response),
            {ok, Added};
        {error, Error} ->
            {error, Sum}
    end,
    gen_tcp:close(Client),
    Updated.
```

Name:_____

6.3.2 better performance [3 points*]

In the solution to the previous question you will (if you did a simple solution) only be able to handle one request at a time. Assuem that we want to handel several requests concurrently, how whould the solution then look like? Don't rewrite everything but describe, using code, what changes needs to be done.

Answer:

```
            Counter = spawn(fun() -> counter(0) end),
            handler(Listen, Counter),
               :

handler(Listen, Counter) ->
    case gen_tcp:accept(Listen) of
        {ok, Client} ->
            spawn(fun() -> request(Client, Counter) end),
            handler(Listen, Counter);
        {error, _} ->
            error
     end.

request(Client, Counter) ->
    case gen_tcp:recv(Client, 0) of
        {ok, Str} ->
            Response = case parse(Str) of
                           {add, N} ->
                               Counter ! {add, N},
                               reply("received " ++ integer_to_list(N));
                           sum ->
                               Counter ! {req, self()},
                               receive
                                 {sum, Sum} ->
                                       reply("the sum is " ++ integer_to_list(Sum))
                               end;
                           error ->
                               notfound()
                       end,
            gen_tcp:send(Client, Response);
        {error, Error} ->
             error
    end,
    gen_tcp:close(Client).

counter(Sum) ->
```

Name:_____

```
receive
  {add, N} ->
     counter(Sum+N);
  {req, Pid} ->
     Pid ! {sum, Sum},
     counter(Sum)
end.
```

Name:_____

6.3.3 double the sum if less than 20 [3 points*]

Assuem we have a solution to the previous question, how would you solve the follwing task?

We want to provide a *atmic operation* that doubles the sum if it is below 20. One might think that this is simply solved by a client by fisrt sending in a request to read the sum, check if it is below 20 and in that case send in a reuest to add an equal amount. The problem with this solution is that it is not atomic, if two clients perform the operation at the same time, they wil ... - yes, what will happen?What is the problem and what do we mean by atomic?

Answer:

Answer:

```
    :
double ->
    Counter ! double_if_less_than_20,
    reply("doubled if less then 20");
  :

  :
double_if_less_than_20 ->
    if
      Sum < 20 ->
         counter(2*Sum);
      true ->
         counter(Sum)
    end;
 :
```