2 Compiler Construction (jdy22)

Here is an OCaml definition of the Ackermann function, `ack`:

```
let rec ack m n =
   if m = 0 then n+1
   else if n = 0 then ack (m-1) 1
   else ack (m-1) (ack m (n-1))
```

(*a*) You would like to run `ack` on an old system with limited stack space and no support for closures, and decide to rewrite it in stages.

(*i*) Rewrite the `ack` function to produce a function `ack_cps` in continuation-passing style so that the function

```
let ack_1 m n = ack_cps m n (fun x -> x)
```

produces the same results as the function `ack`. [5 marks]

(*ii*) Eliminate higher-order functions from your answer to Part (*a*)(*i*) by rewriting `ack_cps` as a function `ack_cps_dfn` in *defunctionalized* form so that the function

```
let ack_2 m n = ack_cps_dfn m n ID
```

produces the same results as the function `ack`. [5 marks]

(*iii*) Convert your answer from Part (*a*)(*ii*) to a function `ack_cps_dfn_list` that uses standard lists rather than custom data types [5 marks]

(*b*) Briefly comment on the way that `ack` and the transformed implementations in Parts (*a*)(*i*), (*a*)(*ii*) and (*a*)(*iii*) use memory, making reference to the stack and heap. [5 marks]