4 Compiler Construction (TGG)

This question concerns the run-time call stack.

(*a*) What is a *run-time stack* and why is it important to a compiler writer? [3 marks]

(*b*) The implementation of a run-time call stack typically uses a *stack pointer* and a *frame pointer*. What are their roles and why do we need two pointers? [3 marks]

(*c*) For some compilers the activation records (stack frames) contain *static links*. What problem are static links used to solve and how do they solve this problem? [3 marks]

(*d*) (*i*) Consider a programming language that does not allow functions to be returned as results, but does allow the nesting of function declarations. Using ML-like syntax, we have the following code in this language.

```
let fun f(x) =
    let
        fun h(k) = k * x

        fun g(z) = h(x + z + 1)
    in
       g(x + 1)
    end
in
    f(17)
end
```

Draw a diagram illustrating the call stack from the call of `f` up to and including the call of function `h`. Make sure all function arguments are included in the diagram and clearly indicate static links. [5 marks]

(*ii*) Using your diagram, explain how the code generated from the body of function `h` can access the values associated with the variables `k` and `x`. In each case make it clear what information is known at compile-time and what information is computed at run-time. [6 marks]