1. x := 1; w := 0
   z := 0
   loop \( x \) \{
       w := z
       z := z + 1
   return w

2. Because no instruction can increase any register by more than 1, and hence at least \( g(x) = x \) instructions will have to be executed to put the value \( g(x) \) in the output register.

3. \( f_1(x) = \begin{cases} 1 & \text{if } x = 0 \\ z & \text{otherwise} \end{cases} \)

   Alternatively \( f = (2x - 1) + 1 \)

4. We observe that any increasing function \( f: \mathbb{N} \to \mathbb{N} \) with \( f(0) > 0 \) is expansive. For suppose, inductively, that \( f(x) > x \). Then since \( x + 1 > x \), \( f(x+1) \geq f(x) + 1 > x + 1 \), as required.

   Observe also that, for all \( n \), \( f_n(0) = 1 > 0 \).

   We show by induction on \( n \) that \( f_n \) is increasing (hence expansive). Evidently, \( f_0 \) is increasing. Suppose \( f_n \) is increasing (hence expansive). Then

   \( f_{n+1}(x+1) = f_n(1) = f_n(f_n(1)) > f_n(1) = f_n(x) \). Hence \( f_{n+1} \) is increasing, and therefore expansive.
5. It suffices to show by induction that \( P_{n+1}(x) \geq P_n(x) \).

For \( n = 0 \), it is obvious that \( P_0(x) \geq P_0(x) \).

Assuming the result for \( n \), we have:

\[
F_{n+2}(x) = \underbrace{\cdots}_n \underbrace{\left( P_{n+1}(x) \right)}_n \\
\geq \underbrace{\cdots}_n \underbrace{\left( P_n(1) \right)}_n \\
= F_{n+1}(x). \quad \text{(since } P_{n+1} \text{ increases)}
\]

Assume first that \( x > 0 \)

Now for \( n \geq 1 \)

\[
F_n(x) = F_n \left( F_n^{(k)}(x) \right) \\
\geq F_{n+1} \left( F_n^{(k)}(x) \right) \\
\geq 2 F_n^{(k)}(x) \quad \text{(using } F_n^{(k)}(x) \geq x > 0) \\
\geq F_n(x) + x \quad \text{since } F_n \text{ is expansive}
\]

On the other hand, if \( x = 0 \), the result follows immediately from the fact that \( F_n \) is expansive.
6. \[ F_1(x) = \begin{cases} 1 & \text{if } x = 0 \\ 2x & \text{otherwise} \end{cases} \]

\[ = \left( 2x \div 1 \right) + 1 \]

This can be computed by:

```
loop x \{ 
  x := x + 3 
  w_1 := 0 
  z := 0 
  loop (x) \{ 
    w_1 := z 
    z := z + 3 
    w_1 := w_1 + 3 
    return w_1 
  \} \}
```

double \( x \) 

as in Q.1

compute \( x := 1 \) 

as in Q.1

and so \( w \) is in \( E_1 \). Suppose \((P_n; \text{return } w_n)\) is a program computing \( F_n(x) \) and using registers \( w_1, \ldots, w_n, y \), putting the return value in \( w_n \). To compute \( F_{n+1} \), execute the program \((P_{n+1}; \text{return } w_{n+1})\) where \( P_{n+1} \) is:

\[ \begin{align*}
  y &:= 1 \\
  w_{n+1} &:= 1 \\
  loop (x) \{ 
    x := y \\
    P_n \{ 
      y := w_n \\
    \} \\
    w_{n+1} := y \\
  \} \\
\end{align*} \]

If \( P_n \) is in \( E_{n-1} \), then \( P_{n+1} \) is in \( E_{n+1} \).
If the running time of $P$ is bounded by $f_\infty (x)$, then the function $g$ computed by $P$ is bounded by $f_n^{(k)}(x)$ and hence by $f_n^{(k+1)}(x)$ for $n \geq 1$. But then $g = f_{n+1}$ cannot be computed by a program with at most $n$ nested loops, since $f_{n+1}$ eventually dominates $f_n^{(k+1)}$. For $n = 0$, we simply observe that $P$ can increase $x$ by only a constant amount, hence $f_1 \neq f_0$. 