1.

$$\begin{array}{rcl}(\lambda x.x\ (x\ y))(\lambda u.u) & \rightarrow & (\lambda u.u)\ ((\lambda u.u)\ y) \\ & \rightarrow & (\lambda u.u)\ y \\ & \rightarrow & y\end{array}$$

2. (a) $id = \lambda x.x$
   (b) $compose = \lambda f.\lambda g.\lambda x.f\ (g\ x)$
   (c) $f = \lambda g.\lambda n.\textbf{if}\ n = 0\ \textbf{then}\ 1\ \textbf{else}\ (fib\ (n-1))\ +\ (fib\ (n-2))$
   $fib = fix\ f$

3. We can no longer write a simple identity function as before, since we need to give a *single* explicit type to its parameter. Thus, we have: $id_{Bool} = \lambda x : Boolean.x$ and $id_{Int} = \lambda x : Integer.x$, and so on for functions. We actually need an infinitude of functions to represent the single functions *id* and *compose* from before!

4. For simplicity's sake, let $U = T \rightarrow T$. We won't show the rules for well-formedness of environments or types, or the rule (Val x). See the last page for the full derivation (Sorry about the readability).

5. (a) $id = \lambda A\ .\ \lambda x : A\ .\ x$
   The type is: $\forall\alpha\ .\ \alpha \rightarrow \alpha$
   (b) $compose = \lambda A\ .\ \lambda B\ .\ \lambda C\ .\ \lambda f : (B \rightarrow C)\ .\ \lambda g : (A \rightarrow B)\ .\ \lambda x : A\ .\ f\ (g\ x)$
   The type is: $\forall\alpha\ .\ \forall\beta\ .\ \forall\gamma\ .\ (\beta \rightarrow \gamma) \rightarrow (\alpha \rightarrow \beta) \rightarrow \alpha \rightarrow \gamma$

6. For both proofs, we proceed by induction on the length of a derivation. That is, for each rule we assume that there is a valid derivation of the premises (everything above the bar), for which our property holds and then show that the property must hold below the bar.

   For progress:

   If $\vdash x : T$ then either x is a value of there is some x' such that $x \rightarrow x'$.

   - Unit rule: if the last rule is the unit rule, then T is $Unit$ and since it only applies when $x = unit$, we have that x is a value.
   - True/False rule: Similar to above; true and false are values.
   - Var rule: $\vdash x : T$ cannot occur, since x is not in the empty context.
   - Abs rule: Similar to Unit Rule; An abstraction is a value.
   - App rule: if t_1 and t_2 are not values, then by the induction hypothesis, $t_1 \rightarrow t_1'$ or $t_2 \rightarrow t_2'$ for some t_1' or t_2'. Otherwise, t_1 must be a λ abstraction, since $t_1 : U \rightarrow T$ (we actually need a small lemma for this step, but we'll assume it for now). Futhermore, $t_2 : U$. This means we have $t_1 = \lambda x : U.e : T$, for some x, e. Thus, we can perform a beta reduction to yield $[x \mapsto t_2]e$. This is our step.
   - If rule: if M is not a value then by the induction hypothesis we can reduce it to some M'.
   - Otherwise it's a Boolean, and it must be *true* or $false$. In either case, we can perform a reduction.

   For preservation (note that here we use induction on the derivation of the reduction, rather than the type of e). We have one case for each reduction rule:

   If e is well typed and $e \rightarrow e'$ then e' is well-typed.

   - The value rules are all vacuous.

- App1 Rule: By the induction hypothesis, the type of e is the type of e', so application has the same type.
- App2 Rule: Again, by the induction hypothesis, e and e' have the same type, so the application has the same type.
- AppAbs Rule: Here we use the substitution property (substituting a term by another term of the same type in some larger term doesn't change the type of the larger term), and the induction hypothesis.
- If Rule: The only way to reduce the if statement results in another if with the same branches (and thus the same type) or a branch (which has the same type as the statement).

$$
\dfrac{
  \dfrac{
    \dfrac{
      y : U, x : (U \to U) \vdash x : (U \to U)
      \qquad
      \dfrac{
        y : U, x : (U \to U) \vdash x : (U \to U)
        \qquad
        y : U, x : (U \to U) \vdash y : U
      }{
        y : U, x : (U \to U) \vdash (xy) : U
      }\,\text{Val Apply}
    }{
      y : U, x : (U \to U) \vdash x(xy) : U
    }\,\text{Val Appl}
  }{
    y : U \vdash \lambda x : (U \to U).x(xy) : (U \to U) \to U
  }\,\text{Val Fun}
  \qquad
  \dfrac{
    y : U, u : U \vdash u : U
  }{
    y : U \vdash \lambda u : U.u : (U \to U)
  }\,\text{Val Fun}
}{
  y : U \vdash (\lambda x : (U \to U).x(xy))(\lambda u : U.u) : U
}\,\text{Val Appl}
$$