Solutions to problems labeled “CL” are in the textbook (Cori and Lascar).

1. CL 5.7
2. CL 5.8
3. CL 5.9
4. CL 5.10
5. CL 5.19

6. We will show that every function computable via a register machine is computable via a
   Turing machine. Suppose the given register machine has \( n \) registers \( R_1, \ldots, R_n \) and \( k \) nodes
   \( n_1, \ldots, n_k \). It will be easiest to use a multi-tape Turing machine (which will be sufficient).
   Our Turing machine will have \( n \) tapes; each tape will be used to simulate one of the registers
   of the register machine, in the sense that tape \( i \) will contain an initial string of \( m \) 1’s when
   register \( R_i \) contains the number \( m \). The standard starting configuration of the multi-tape
   machine is already in accord with this, so there is no additional setup necessary.

   We will use a group of states to simulate each node of the register machine. The group will
   move the machine to the left of the tapes, and if the node it is simulating refers to register \( R_i \),
   the group will focus on tape \( i \). If the node is to add 1 to \( R_i \), then the machine moves to the
   right edge of the initial segment of 1’s and writes an additional 1 at the right end, and then
   moves to the group of states corresponding to the new node. If the node was a subtracting
   node, then if the machine reads a blank at the beginning of the tape it moves to the group
   of states corresponding to the node branched to when \( R_i \) was empty; otherwise it erases the
   rightmost 1 on the tape and then goes to the appropriate group of states.

   All that is left to do is to clean up the state of the tapes once the register machine reaches
   its halt node.

   Note: The converse of this exercise holds as well. Every recursive function is computable via
   a register machine (with \( p \)-ary functions computed in the analogous way to unary ones given
   here).