14.A From Textbook 3.3 6ac

6a $a_n = 4n - 2$
$\forall n \in \mathbb{N}\ a_{n+1} - a_n = 4(n+1) - 2 - (4n-2) = 4 \longrightarrow \begin{cases} a_1 = 2 \\ a_{n+1} = a_n + 4 \end{cases}$

6c $b_n = n(n+1)$
$\forall n \in \mathbb{N}\ b_{n+1} - b_n = (n+1)(n+2) - n(n+1) = 2(n+1) \longrightarrow \begin{cases} b_1 = 2 \\ b_{n+1} = b_n + 2(n+1) \end{cases}$

There are many other solutions, for example $\begin{cases} b_1 = 2 \\ b_2 = 6 \\ b_{n+1} = 2(b_n + 1) - b_{n-1} \end{cases}$

14.B From Textbook 3.3 10

We have $\begin{cases} f_0 = 0 \\ f_1 = 1 \\ f_{n+1} = f_n + f_{n-1} \end{cases}$ and we are looking for $S_n = f_1^2 + f_2^2 + f_3^2 + \cdots + f_n^2 = \sum_{i=1}^{n} f_i^2$

The formula for $S_n = f_n f_{n+1}$ is true for $n = 1$.
Assume $S_n = f_n f_{n+1}$ is true for a given n, then

$$\begin{aligned} S_{n+1} &= \sum_{i=1}^{n+1} f_i^2 \\ &= \sum_{i=1}^{n} f_i^2 + f_{n+1}^2 \\ &= S_n + f_{n+1}^2 && \text{(by definition of } S_n\text{)} \\ &= f_n f_{n+1} + f_{n+1}^2 && \text{(Induction hypothesis)} \\ &= f_{n+1}(f_n + f_{n+1}) \\ &= f_{n+1} f_{n+2} && \text{(by definition of } f_{n+2}\text{)} \end{aligned}$$

The formula is true for $n+1$. Therefore we have proven by induction that $\sum_{i=1}^{n} f_i^2 = f_n f_{n+1}$ holds for every positive integer n.

14.C From Textbook 3.3 30

A palindrome is a string that reads the same backward as it does forward.
Let S be the set of the bit strings which are palindromes.

$$S: \begin{cases} \lambda \in S \text{ (the empty string)} \\ 0 \in S \text{ and } 1 \in S \\ \text{if } x \in S \text{ then } 0x0 \in S \text{ and } 1x1 \in S. \end{cases}$$

It is up to you to decide if the empty string λ is in S or not. If it is not, you have to add the strings 00 and 11 in S.

15.A From Textbook 3.4 2

```
**procedure** *sum*($n$:nonnegative integer)
**if** $n=0$ then $sum(0) := 0$
**else** $sum(n) := n + sum(n-1)$
```

15.B From Textbook 3.4 8

For the purposes of our algorithm, we will treat a list as an indexed set (i.e. a set where each element has a corresponding index).
The procedure will return a pair: the first component of the pair is a mode of the list, the second is the number of times that mode occurs.
If the list is one element, we return the pair consisting of that one element and 1.
Otherwise, we remove the last element a_n from the list (as well as any other occurrences of that same element in the list). If a_n was the only element in the list, we return it and the length of the list.
We recurse on the list with the a_n removed (and all occurrences); if the answer m has a higher number of occurrences than the number of occurrences of a_n, we return m as the mode, otherwise we return the last element.

```
procedure mode($L = \{a_1, \ldots, a_n\}$ : list of integers)
$L' := L - \{a_i | a_i = a_n\}$
$k := |L - L'|$
if $k = n$ then
    return $(a_n, n)$
else
    $(m, t) :=$ mode($L'$)
    if $t > k$ then return $(m, t)$
    else return $(a_n, k)$
```

16.A From Textbook 3.5 2

$$\frac{\begin{array}{c}(\text{True} \wedge (x < 0))\ \{x := 0\}\ (x \geq 0) \\ (\text{True} \wedge \neg(x < 0)) \ \rightarrow\ (x \geq 0)\end{array}}{(\text{True})\ \{\textbf{if } x < 0 \textbf{ then } x := 0\}\ (x \geq 0)}$$

Termination is obvious. The first premise trivially holds. The second is shown by distributing the negation over the ordering relation: $\neg(x < 0) \iff (x \geq 0)$.

16.B From Textbook 3.5 4

I presume there is a typo in the text. We'll end up with

$$q \;\equiv\; \big((x < y) \;\wedge\; (\min = x)\big) \;\vee\; \big((x \geq y) \;\wedge\; (\min = y)\big)$$

at the end of the rule for if then else. The text asks for

$$q' \;\equiv\; \big((x \leq y) \;\wedge\; (\min = x)\big) \;\vee\; \big((x > y) \;\wedge\; (\min = y)\big)$$

Nevertheless, those two are equivalent: if $x \neq y$ then they are equivalent. If $x = y$, then max is set to y instead of x, but $x = y$ so it doesn't make any difference.

Proof:
(True $\wedge$ $(x < y)$) $\{\min := x\}$ $\big((x < y) \;\wedge\; (\min = x)\big)$.
This is obvious.
(True $\wedge$ $\neg(x < y)$) $\{\min := y\}$ $\big((x \geq y) \;\wedge\; (\min = y)\big)$.
This is true also because $\neg(x < y) \iff (x \geq y)$.
We have

$$\begin{array}{lcl} A &\equiv& \big((x < y) \wedge (\min = x)\big) \;\rightarrow\; \big((x < y) \wedge (\min = x)\big) \vee \big((x \geq y) \wedge (\min = y)\big) \;\equiv\; q \\ B &\equiv& \big((x \geq y) \wedge (\min = y)\big) \;\rightarrow\; \big((x \geq y) \wedge (\min = y)\big) \vee \big((x < y) \wedge (\min = x)\big) \;\equiv\; q \end{array}$$

Therefore the program is correct (does terminate), the final assertions q and q' being equivalent.

16.C From Textbook 3.5 12

In the following, the symbol `I` is shorthand for . . .

```
  a > 0 and d > 0 and a, d, q, r are integers
```

We begin our fairly formal proof by showing that `I and a = dq + r and r >= 0` is an invariant of the loop. [`(A)`–`(F)` below are Hoare triples `P{S}Q`, written vertically.]

```
(A)
I and a = dq + r and r >= 0 and r >= d
{ r := r - d }
I and a = d(q + 1) + r and r >= 0
  [initially, a = dq + r = dq + d + r - d = d(q + 1) + (r - d)
   and r - d >= 0, so assigning r - d to r leaves us with
   a = d(q + 1) + r and r >= 0]

(B)
I and a = d(q + 1) + r and r >= 0
{ q := q + 1 }
I and a = dq + r and r >= 0
```

```
(C)
I and a = dq + r and r >= 0 and r >= d
{ r := r - d; q := q + 1 }
I and a = dq + r and r >= 0
  [by the inference rule for ;, using the Hoare triples (A) and (B) above.
   note that the initial and final assertions do match correctly, e.g., the
   initial assertion of (B) coincides with the final assertion of (A), etc.]
```

We now use `(C)` to prove that the core of the algorithm is partially correct.

```
(D)
I and a = dq + r and r >= 0
{while r >= d
begin
    r := r - d;
    q := q + 1
end}
I and a = dq + r and r >= 0 and r < d
  [by the inference rule for while, using (C), i.e., using the fact that
   I and a = dq + r and r >= 0 is a loop invariant]
```

Finally, we show that the initialization `r := a; q := 0` does indeed establish the precondition that we assumed above; from there we easily derive the partial correctness of the algorithm as a whole.

```
(E)
a > 0 and d > 0 and a, d are integers
{r := a; q := 0}
I and a = dq + r and r >= 0
  [a = dq + r holds simply because dq = 0 and r = a]

(F)
a > 0 and d > 0 and a, d are integers
{r := a; q := 0;
while r >= d
begin
    r := r - d;
    q := q + 1
end}
I and a = dq + r and r >= 0 and r < d
  [by the inference rule for ;, using (E) and (D)]
```