Programming Language Comprehensive, September 1995
Answers

Question 1:

```
{n>=m}
x:=0;
i := m;
{x = number of zeroes in interval b[m]...b[i-1] and i<=(n+1)}  Invariant
while i <= n do
begin
  if b[i] = 0 then x:=x+1;
  i:=i+1;
end
{x = number of zeroes in interval b[m]...b[n]}
```

The and part of the invariant is crucial for deriving the postcondition from the invariant and (not $i <= n$), which comes from the boolean test. You must use a little logic, together with weakening the consequent.

Simpler version:

```
{n>=m}
x:=0;
i := m-1;
{x = number of zeroes in interval b[m]...b[i]}  Invariant
while i <> n do
begin
  i:=i+1;
  if b[i] = 0 then x:=x+1;
end
{x = number of zeroes in interval b[m]...b[n]}
```

Now the reasoning follows much more easily.

Question 2: The state refers to the contents of memory, or more specifically to the (current) contents of storage variables referred to by a program. The environment is an association of meanings to identifiers, or in syntactic terms often represented by a symbol table.

The difference is that assignment statements change the state, whereas definitions, such as procedure declarations, change the environment. Furthermore, the change effected by assignment is (generally) irreversible, is persistent, and destructively alters variable contents. In contrast, a local declaration creates a "new" binding for an identifier, but does not destroy a previous binding; the previous value is saved, and sometimes reused later.

In Scheme and Lisp (and certain object-oriented languages) state and environment are confused. Assignment and procedure declarations both change the environment (albeit in different ways). A small caveat is the set-car and set-cdr operators, which are part-way to respecting a state-environment distinction. In Pascal the distinction is for the most part maintained, though one must explain var parameters properly.

Question 3. (A) $\{P[e/x]\}x := e\{P\}$. Some instances in (C).

(B)

```
{P}C{Q}  {Q}E{S}
----------------- Sequencing
  {P} C;E {S}
```

(C)

```
(1) {x*2=10} y:=x*2 {y=10}    Assignment axiom
(2) {(x+1)*2=10} x:=x+1 {x*2=10}   Assignment
(3) {(x+1)*2=10} x:=x+1; y:=x*2 {y=10}   from (1), (2), and Sequencing
(4) {x=4 implies (x+1)*2=10}    Mathematical fact about integers
(5) {x=4}x:=x+1; y:=x*2{y=10}  from (3),(4) and precedent strengthening
```

Question 4 (A) $f_0 = \bot$. $f_{i+1} = (\lambda n.\, if\ n = 0\ then\ 1\ else\ n * (f_i(n-1))$.

(B) The type is $(t \to t) \to t$, for arbitrary t. We can write

$$(Y_{\text{integer}\to\text{integer}}(\lambda fact\,.\,\lambda n\ if\ n = 0\ then\ 1\ else\ n * (fact(n-1))$$

The type of Y is in the subscript.

(C) $Y(f) = \sqsubseteq_{i\in\omega}\ f^i(\bot)$ where $f_0 = identity$ and $f^{i+1} = f; f^i$.

Question 5.

In Pascal, procedure calls are done using a stack. In Scheme or Lisp, the procedure call and parameter allocation is heap-based. The use of heap comes from the confusion between the state and the environment, and because of function-returning-functions. In Lisp, the binding of arguments to formal parameters can be changed on successive calls of a procedure that is produced as a result of a procedure call, so the environment cannot be generally discarded on exit from a block.

Question 6. (A) Regular expressions are used for lexical analysis, and context-free grammars for parsing.

(B) The requested regular expressions are $(a^*b^*)^*$ and $((a^*b^*)^*b(a^*b^*)^*b(a^*b^*)^*)^*$

(C) A context-free grammar is a grammar given by productions of the form $A \to x_1 A_1 ... A_n x_{n+1}$
where each x_i is a sequence of terminal symbols, A and A_i are non-terminals, and in which there is a specified start (non-terminal) symbol. Here is a context-free grammar that has infinitely many strings in its language:

$S \to aAb$
$A \to b \mid aA$

Here is a grammar that is not context free.

$S \to aAb$
$aA \to b \mid cA$

Notice the terminal on the left.