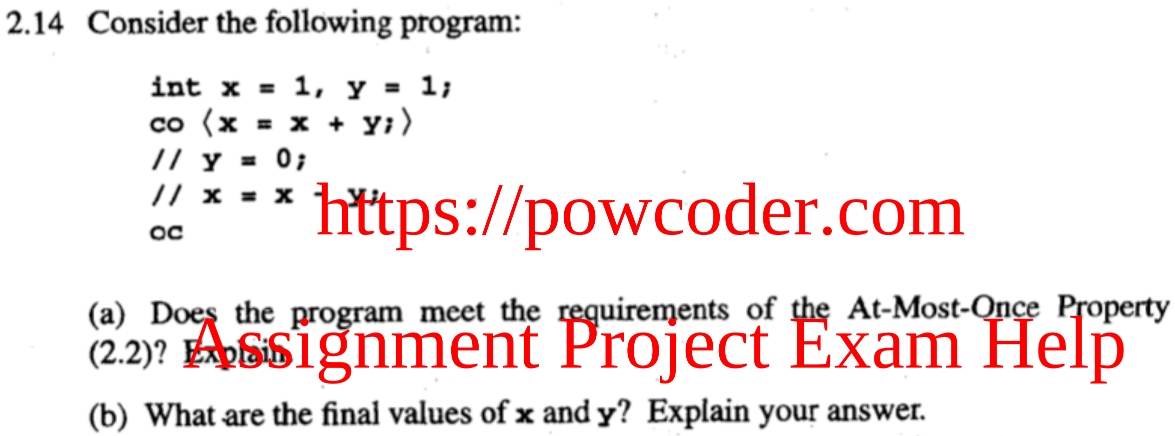
Final Practice Problems

Locks:

Problem 2.14(b) (replace the angle brackets with Lock and Unlock)



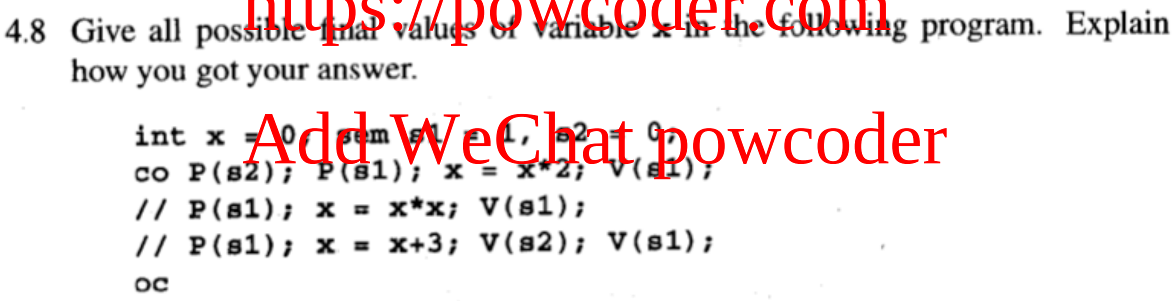
2.14 Consider the following program:

```
int x = 1, y = 1;
co ⟨x = x + y;⟩
// y = 0;
// x = x - y;
oc
```

(a) Does the program meet the requirements of the At-Most-Once Property (2.2)? Explain.

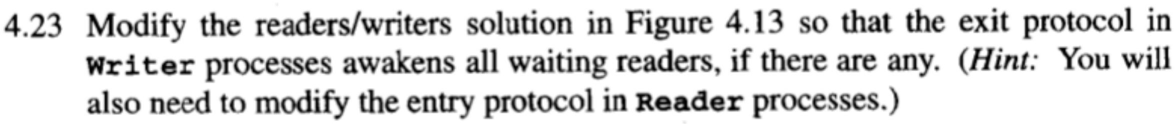
(b) What are the final values of x and y? Explain your answer.

Semaphores:



4.8 Give all possible final values of variable x in the following program. Explain how you got your answer.

```
int x = 0; sem s1 = 1, s2 = 0;
co P(s2); P(s1); x = x*2; V(s1);
// P(s1); x = x*x; V(s1);
// P(s1); x = x+3; V(s2); V(s1);
oc
```



4.23 Modify the readers/writers solution in Figure 4.13 so that the exit protocol in Writer processes awakens all waiting readers, if there are any. (*Hint:* You will also need to modify the entry protocol in Reader processes.)

Figure 4.13 is on next page.

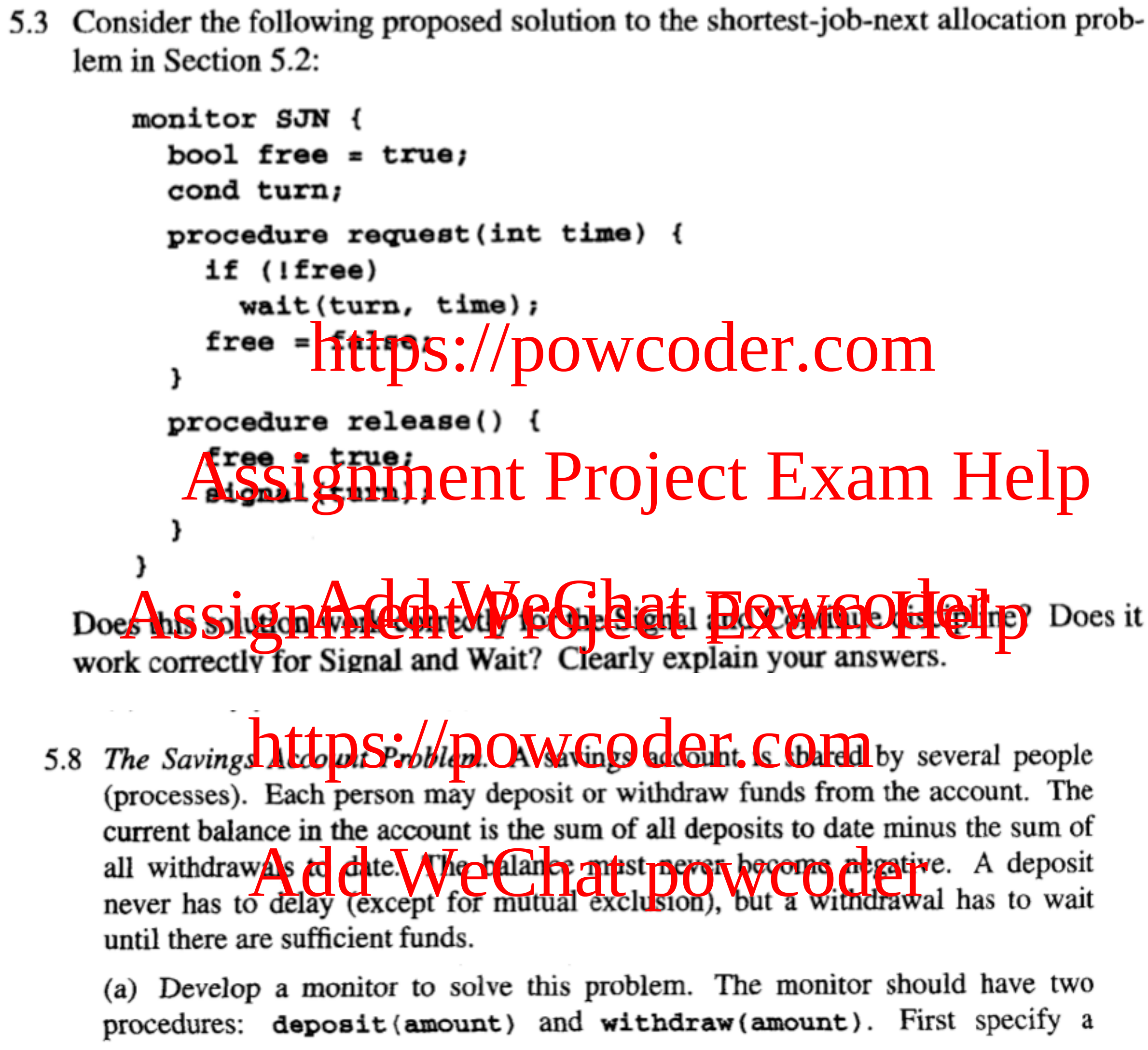
```
int nr = 0,     ## RW: (nr==0 or nw==0) and nw<=1
    nw = 0;
sem e = 1,      # controls entry to critical sections
    r = 0,      # used to delay readers
    w = 0;      # used to delay writers
                # at all times 0 <= (e+r+w) <= 1
int dr = 0,     # number of delayed readers
    dw = 0;     # number of delayed writers

process Reader[i = 1 to M] {
  while (true) {
    # ⟨await (nw == 0) nr = nr+1;⟩
      P(e);
      if (nw > 0) { dr = dr+1; V(e); P(r); }
      nr = nr+1;
      if (dr > 0) { dr = dr-1; V(r); }
      else V(e);
    read the database;
    # ⟨nr = nr-1;⟩
      P(e);
      nr = nr-1;
      if (nr == 0 and dw > 0) { dw = dw-1; V(w); }
      else V(e);
  }
}

process Writer[j = 1 to N] {
  while (true) {
    # ⟨await (nr == 0 and nw == 0) nw = nw+1;⟩
      P(e);
      if (nr > 0 or nw > 0) { dw = dw+1; V(e); P(w); }
      nw = nw+1;
      V(e);
    write the database;
    # ⟨nw = nw-1;⟩
      P(e);
      nw = nw-1;
      if (dr > 0) { dr = dr-1; V(r); }
      elseif (dw > 0) { dw = dw-1; V(w); }
      else V(e);
  }
}
```

Figure 4.13 A readers/writers solution using passing the baton.

Monitors



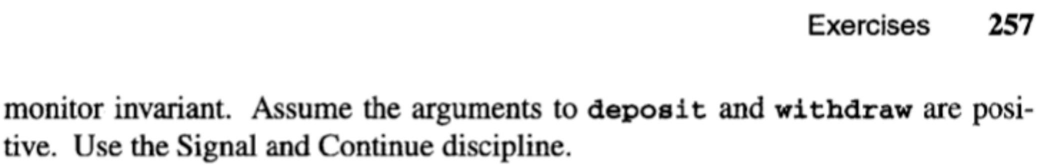
5.3 Consider the following proposed solution to the shortest-job-next allocation problem in Section 5.2:

```
monitor SJN {
  bool free = true;
  cond turn;
  procedure request(int time) {
    if (!free)
      wait(turn, time);
    free = false;
  }
  procedure release() {
    free = true;
    signal(turn);
  }
}
```

Does this solution work correctly for the Signal and Continue discipline? Does it work correctly for Signal and Wait? Clearly explain your answers.

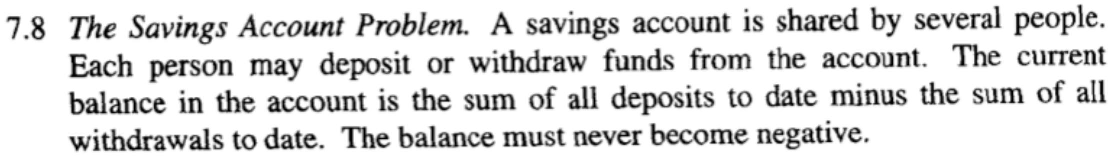
5.8 *The Savings Account Problem.* A savings account is shared by several people (processes). Each person may deposit or withdraw funds from the account. The current balance in the account is the sum of all deposits to date minus the sum of all withdrawals to date. The balance must never become negative. A deposit never has to delay (except for mutual exclusion), but a withdrawal has to wait until there are sufficient funds.

(a) Develop a monitor to solve this problem. The monitor should have two procedures: `deposit(amount)` and `withdraw(amount)`. First specify a

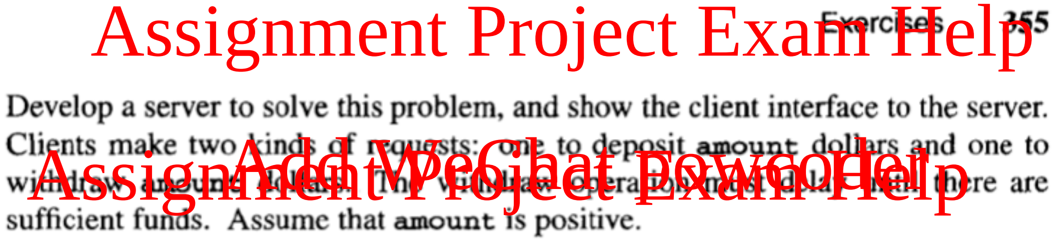


monitor invariant. Assume the arguments to `deposit` and `withdraw` are positive. Use the Signal and Continue discipline.

Message Passing



7.8 *The Savings Account Problem.* A savings account is shared by several people. Each person may deposit or withdraw funds from the account. The current balance in the account is the sum of all deposits to date minus the sum of all withdrawals to date. The balance must never become negative.



Develop a server to solve this problem, and show the client interface to the server. Clients make two kinds of requests: one to deposit amount dollars and one to withdraw amount dollars. The withdraw operation must delay until there are sufficient funds. Assume that amount is positive.