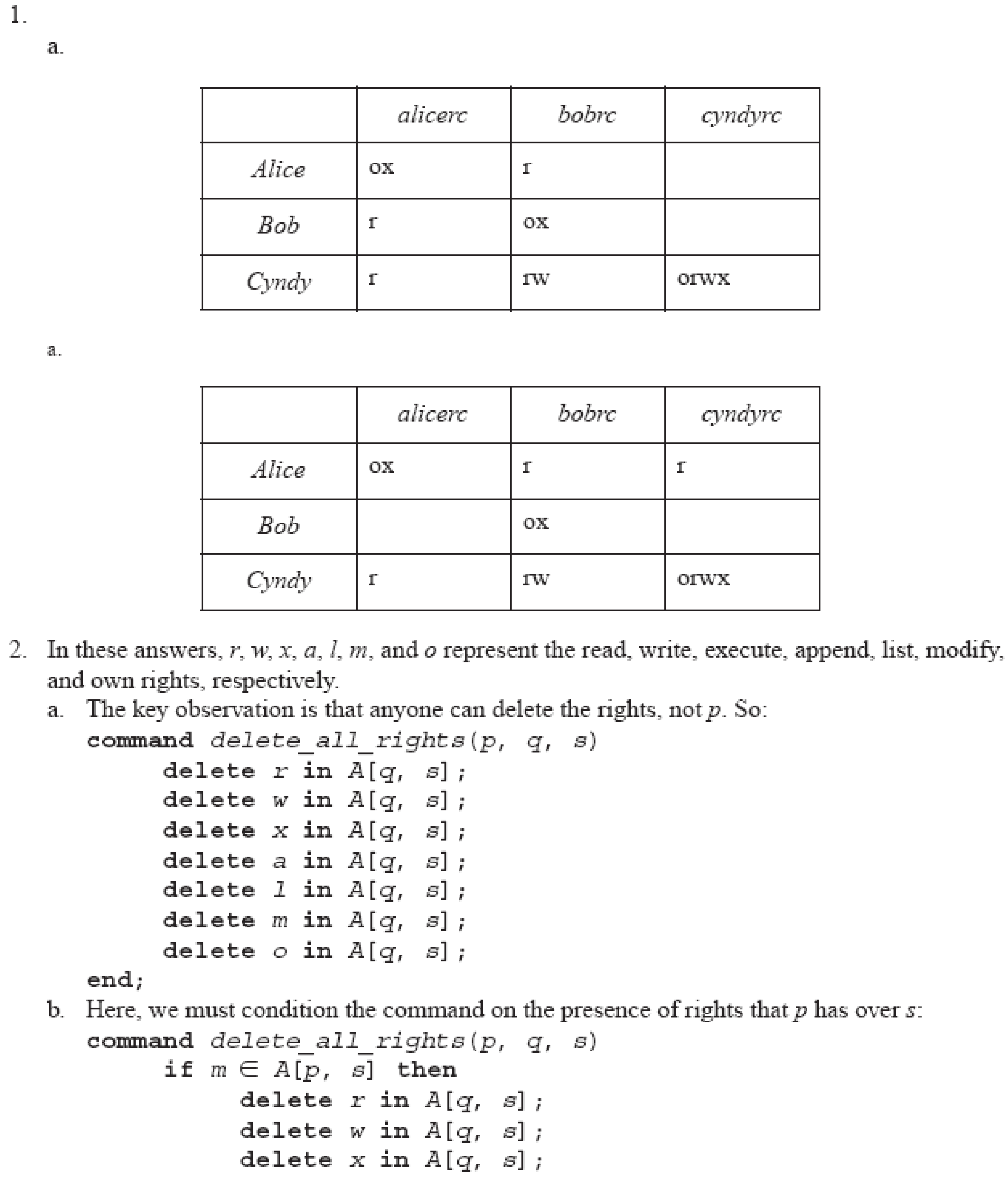
IS2150/TEL2810 Introduction to Security

Homework 3 Sample Solution

Question 1: Section 2.6 Exercises 1 and 2



1.

a.

	alicerc	*bobrc*	*cyndyrc*
Alice	ox	r	
Bob	r	ox	
Cyndy	r	rw	orwx

a.

	alicerc	*bobrc*	*cyndyrc*
Alice	ox	r	r
Bob		ox	
Cyndy	r	rw	orwx

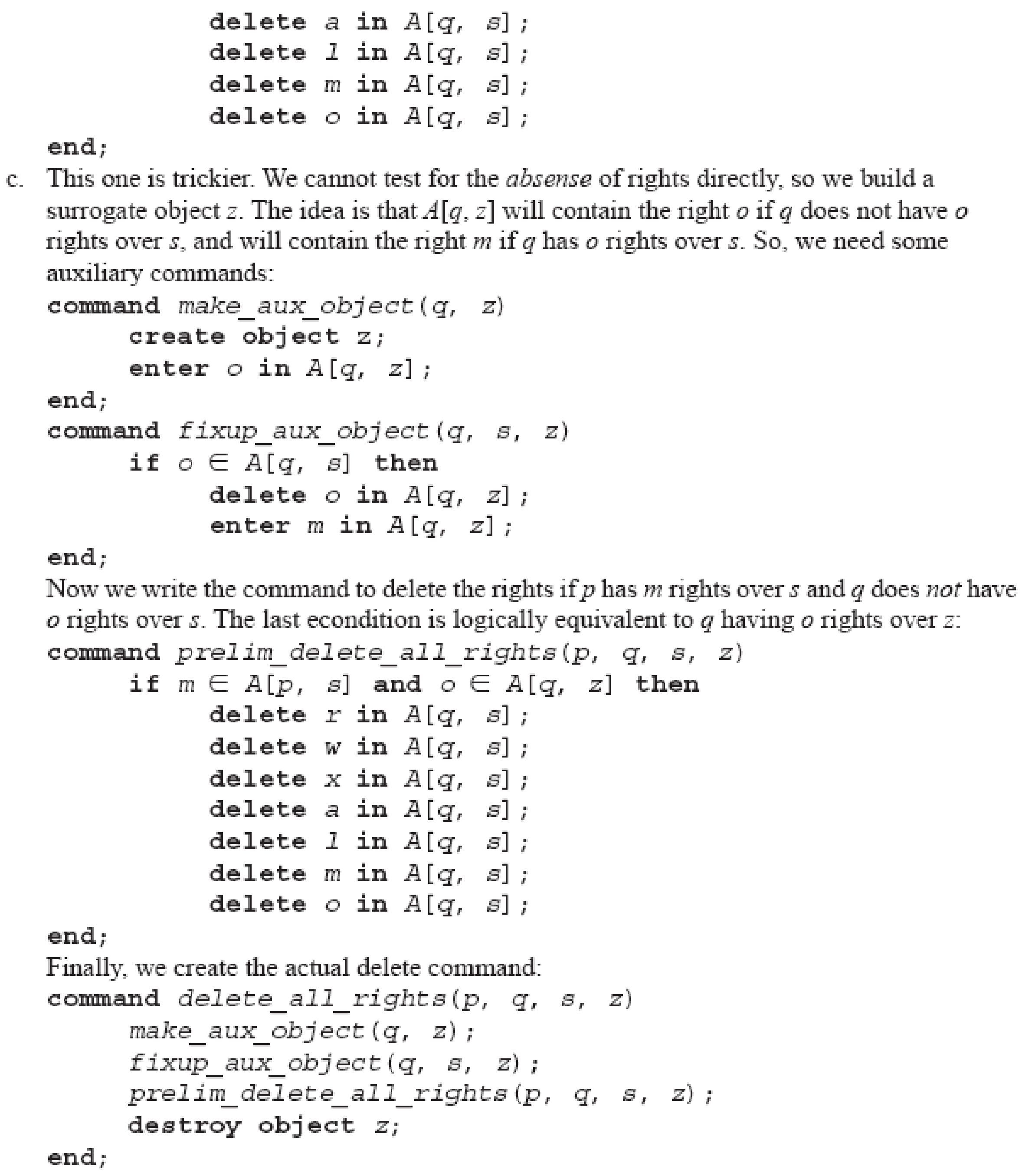
2. In these answers, *r*, *w*, *x*, *a*, *l*, *m*, and *o* represent the read, write, execute, append, list, modify, and own rights, respectively.

a. The key observation is that anyone can delete the rights, not *p*. So:

```
command delete_all_rights(p, q, s)
          delete r in A[q, s];
          delete w in A[q, s];
          delete x in A[q, s];
          delete a in A[q, s];
          delete l in A[q, s];
          delete m in A[q, s];
          delete o in A[q, s];
end;
```

b. Here, we must condition the command on the presence of rights that *p* has over *s*:

```
command delete_all_rights(p, q, s)
          if m ∈ A[p, s] then
                    delete r in A[q, s];
                    delete w in A[q, s];
                    delete x in A[q, s];
```



```
            delete a in A[q, s];
            delete l in A[q, s];
            delete m in A[q, s];
            delete o in A[q, s];
end;
```

c. This one is trickier. We cannot test for the *absense* of rights directly, so we build a surrogate object *z*. The idea is that *A*[*q*, *z*] will contain the right *o* if *q* does not have *o* rights over *s*, and will contain the right *m* if *q* has *o* rights over *s*. So, we need some auxiliary commands:

```
command make_aux_object(q, z)
      create object z;
      enter o in A[q, z];
end;
command fixup_aux_object(q, s, z)
      if o ∈ A[q, s] then
            delete o in A[q, z];
            enter m in A[q, z];
end;
```

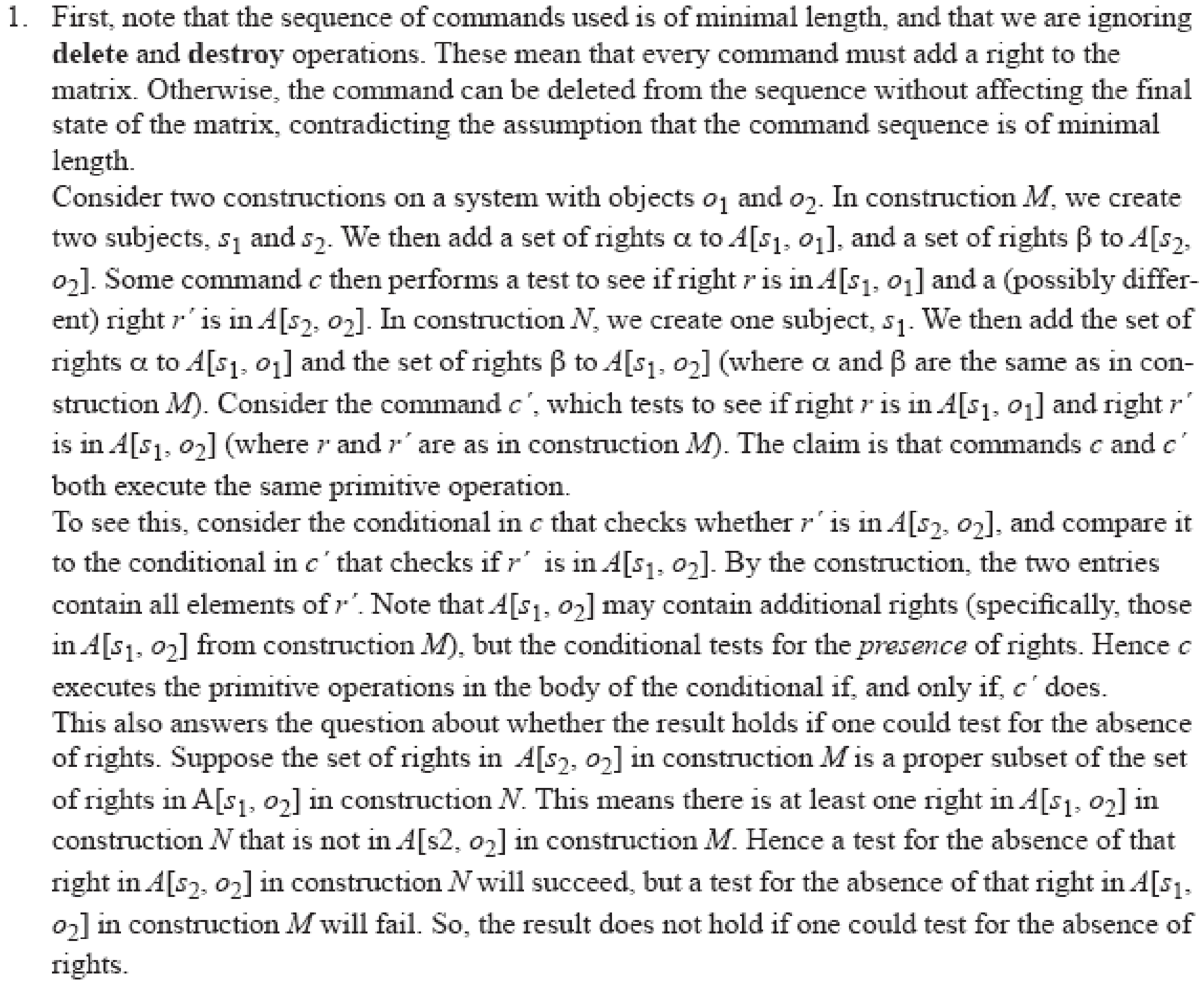
Now we write the command to delete the rights if *p* has *m* rights over *s* and *q* does *not* have *o* rights over *s*. The last econdition is logically equivalent to *q* having *o* rights over *z*:

```
command prelim_delete_all_rights(p, q, s, z)
      if m ∈ A[p, s] and o ∈ A[q, z] then
            delete r in A[q, s];
            delete w in A[q, s];
            delete x in A[q, s];
            delete a in A[q, s];
            delete l in A[q, s];
            delete m in A[q, s];
            delete o in A[q, s];
end;
```

Finally, we create the actual delete command:

```
command delete_all_rights(p, q, s, z)
      make_aux_object(q, z);
      fixup_aux_object(q, s, z);
      prelim_delete_all_rights(p, q, s, z);
      destroy object z;
end;
```

Question 1: Section 3.5 Exercise 1

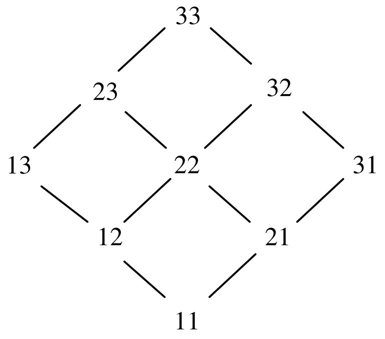


1. First, note that the sequence of commands used is of minimal length, and that we are ignoring **delete** and **destroy** operations. These mean that every command must add a right to the matrix. Otherwise, the command can be deleted from the sequence without affecting the final state of the matrix, contradicting the assumption that the command sequence is of minimal length.
   Consider two constructions on a system with objects o_1 and o_2. In construction M, we create two subjects, s_1 and s_2. We then add a set of rights α to $A[s_1, o_1]$, and a set of rights β to $A[s_2, o_2]$. Some command c then performs a test to see if right r is in $A[s_1, o_1]$ and a (possibly different) right r' is in $A[s_2, o_2]$. In construction N, we create one subject, s_1. We then add the set of rights α to $A[s_1, o_1]$ and the set of rights β to $A[s_1, o_2]$ (where α and β are the same as in construction M). Consider the command c', which tests to see if right r is in $A[s_1, o_1]$ and right r' is in $A[s_1, o_2]$ (where r and r' are as in construction M). The claim is that commands c and c' both execute the same primitive operation.
   To see this, consider the conditional in c that checks whether r' is in $A[s_2, o_2]$, and compare it to the conditional in c' that checks if r' is in $A[s_1, o_2]$. By the construction, the two entries contain all elements of r'. Note that $A[s_1, o_2]$ may contain additional rights (specifically, those in $A[s_1, o_2]$ from construction M), but the conditional tests for the *presence* of rights. Hence c executes the primitive operations in the body of the conditional if, and only if, c' does.
   This also answers the question about whether the result holds if one could test for the absence of rights. Suppose the set of rights in $A[s_2, o_2]$ in construction M is a proper subset of the set of rights in A[s_1, o_2] in construction N. This means there is at least one right in $A[s_1, o_2]$ in construction N that is not in A[s2, o_2] in construction M. Hence a test for the absence of that right in $A[s_2, o_2]$ in construction N will succeed, but a test for the absence of that right in $A[s_1, o_2]$ in construction M will fail. So, the result does not hold if one could test for the absence of rights.

Question 2: Exercise on Lattice

S = {11, 12, 13, 21, 22, 23, 31, 32, 33}

The relation $\lesssim$ over set S is partial order, as it is reflexive (e.g., 11$\lesssim$11), anti-symmetric (e.g., 11$\lesssim$12 but 12$\not\lesssim$11), and transitive (e.g., 11$\lesssim$12, 12$\lesssim$32, and 11$\lesssim$32). However, it is not a total order because not every pair of elements are comparable (e.g., 13$\not\lesssim$32 and 32$\not\lesssim$13, i.e., the relation is not applying to 13 and 32 either way).



It’s also a lattice. Because in addition to being partial order (reflexive, anti-symmetric, and transitive), every two elements have a greatest lower bound and least upper bound.

Question 3: Section 4.8 Exercise 3, 4, 5, and 6

3. (a) Assume that the system has no integrity controls. This is true. If a system lacks integrity, then data can be changed without restraint. So, anyone can change another user’s authentication information, allowing them access to that user’s account—and allowing them to see any data for that user or, by generalizing this in the obvious way, any user on the system. If some integrity controls work, then the ability of the system to provide confidentiality depends on the effectiveness of the integrity controls and their use to protect critical information.

   b. Assume that the system has no confidentiality controls. If there is no confidentiality, then all authentication information will be available. Unless authentication mechanisms do not use secret information (for example, biometrics or positions), any user can authenticate as another user. Hence there is no integrity. Now suppose authentication information does not rely on confidentiality. Can the data in a file be kept confidential? To do so, either the user must be prevented from reading the file (for which there are no controls) or from reading the data in the file (for example, by cryptography). In the latter case, if the data is encrypted on the system, the key must be available, and as there is no confidentiality, the key can be read. If the data is not encrypted on the system, then the data cannot be used on the system but will remain confidential. So, the answer is that the system cannot provide integrity. But if data is protected when placed on the system, it will remain protected as long as confidentiality mechanisms were applied to the data itself (and not to the containing object) and the mechanisms to undo them are not on the system.

4. The problem with the cryptographer’s claim is how to protect the keys. At some point, the cryptographic keys must be available to encipher, decipher, or validate the integrity of data. If the keys are kept in memory, they must be protected, either by other cryptographic keys (which require similar protection) or by non-cryptographic access control mechanisms. If the keys are kept off-line (for example, in a smart card or a dongle), access to the external unit must be protected either by cryptographic keys (which require the protection discussed earlier) or non-cryptographic access control. By a simple process of induction, or *reductio ad absurdam*, non-cryptographic based access control mechanisms must be used at some point, refuting the cryptographer’s claim.

5. Classify each of the following as an example of a mandatory, discretionary, or originator controlled policy, or a combination thereof. Justify your answers.

   1. The file access control mechanisms of the UNIX operating system
      discretionary access control
      Since users can assign and modify permissions that they possess, access control is discretionary.

   2. A system in which no memorandum can be distributed without the author's consent
      originator access control
      This would be originator access control. This is because if I am the author of the memorandum I am the one who can say my information can be distributed, no one else can.

   3. A military facility in which only generals can enter a particular room
      mandatory access control
      The system controls access and an individual cannot change that. There is a somewhat tricky scenario though that could possibly make this discretionary; if there is an owner of the 'military facility' and this person also had the ability to promote military personnel to 'general'. In this way the facility owner could grant access to their facility.

   4. A university registrar's office, in which a faculty member can see the grades of a particular student provided that the student has given written permission for the faculty member to see them.

discretionary access control
Here the student grants the permission to the faculty to see the grades. If he doesn't grant permission to a particular faculty member, that faculty member can't see the grades.

This is a combination of an originator controlled access control policy and a discretionary access control policy. The originator, which is the registrar, controls dissemination of the data, but the student also has some control, and allows access to the individual record based upon the identity of the faculty member.

Question 4: Section 5.5 Exercise 2, 4, 5, and 6

2. Given the security levels TOP SECRET, SECRET, CONFIDENTIAL, and UNCLASSIFIED (ordered from highest to lowest), and the categories A, B, and C, specify what type of access (read, write, both, or neither) is allowed in each of the following situations. Assume that discretionary access controls allow anyone access unless otherwise specified.

   Simple security property says that a subject can write to object if subject compartment dominates object compartment. *-property says that subject can write to object if object compartment dominates subject compartment. Let (L,C) and (L', C') be compartments for different entities. (*(L,C) dominates (L',C')* $\Leftrightarrow$ *L'* $\leq$ *L* and *C'* $\subseteq$ *C)* is the principle we are going to apply to specify what type of access that the following sentences have.

   a. Paul, cleared for (TOP SECRET, { A, C }), wants to access a document classified (SECRET, { B, C }).
   Paul **cannot read** and **cannot write** to the document because Paul's clearance level does not dominate document's classification level and vice versa.

   b. Anna, cleared for (CONFIDENTIAL, { C }), wants to access a document classified (CONFIDENTIAL, {B }).
   Anna **cannot read** and **cannot write** to the document because Anna Paul's clearance level does not dominate document's classification level and vice versa.

   c. Jesse, cleared for (SECRET, { C }), wants to access a document classified (CONFIDENTIAL, { C }).
   Jesse **can read** document because Jesse Paul's clearance level dominates document 's classification level, but Jesse **cannot write** to the document because document's classification level does not dominate Jesse's clearance level.

   d. Sammi, cleared for (TOP SECRET, { A, C }), wants to access a document classified (CONFIDENTIAL, {A }).
   Sammi **can read** document because Sammi Paul's clearance level dominates document's classification level, but Sammi **cannot write** to the document because document's classification level does not dominate Jesse's clearance level.

   e. Robin, who has no clearances (and so works at the UNCLASSIFIED level), wants to access a document classified (CONFIDENTIAL, { B }).
   Robin **cannot read** document because Jesse Paul's clearance level does not dominate document's classification level, but Robin **can write** to the document because document's classification level dominates Jesse's clearance level.

4. In the DG/UX system, the virus prevention region is below the user region to prevent any user programs from altering (writing) code or data in a region that contains system or site executables. For example, if a user loads and executes a program in the user region, that program cannot alter system executables because of the rule forbidding writes down. Hence computer viruses can spread only within the user region. Note that the DG/UX system disallows writes up, so the computer virus at the user region could not alter executables or other files in the administrative region. (In a strict implementation of the Bell-LaPadula model, they would be able to write to information in this region.)
5. In the DG/UX system, the administrative region is above the user region to prevent the users from reading information stored at that level. For example, the Identification and Authorization database contains sensitive information, such as authentication information, that users should notbe able to see. Note that the DG/UX system also disallows writes up, so users cannot append to or alter information in this region. (In a strict implementation of the Bell-LaPadula model, they would be able to write to information in this region.)
6. The three properties are the *-property, the simple security condition, and the discretionary security property. The *-property bars writing down; as *raising* the classification of an object effectively writes up, raising the classification does not violate any property.
   The simple security condition bars reading up. Raising the classification makes the object once available to a particular compartment no longer available. So, until the information in that object is changed, a lower-level subject is aware of the contents of a higher-level object. If you interpret the simple security property as barring *any* knowledge of what is in a higher-level object, this is a violation of the simple security condition. However, the lower-level subject cannot know whether a higher-level subject has changed that object. So the simple security condition is not violated if you interpret it to mean that an actual *read* must occur. (This is the customary interpretation, by the way.) An implementation observation: if access is checked only at *open*s (as it is with the UNIX operating system), a violation may arise when an open file is reclassified upwards. The process having the file open can still read it. If that process is at a lower level, each read violates the simple security condition.
   The discretionary security property does not affect reading or writing at different levels.

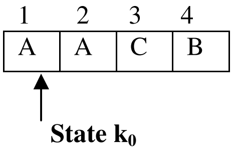
Question 5:

Mappings:

$S = \{s_1, s_2, s_3, s_4\}$
$O = S$
$R = \{\text{own, end, A, B, C}, k_0, k_1, k_2, k_3\}$

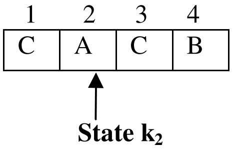
Also, each transition function input/output in TM maps to corresponding commands in a protection system.

Initial state given:



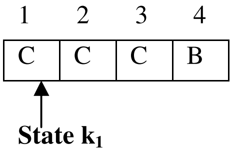
	s_1	s_2	s_3	s_4
s_1	A, k_0	own		
s_2		A	own	
s_3			C	own
s_4				B, end

Applying the transition function δ (k_0,A) = (k_2,C,R) we get:



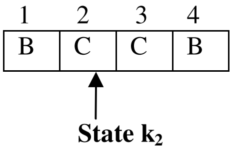
	s_1	s_2	s_3	s_4
s_1	C	own		
s_2		A, k_2	own	
s_3			C	own
s_4				B, end

Applying the transition function δ (k_2,A) = (k_1,C,L) we get :



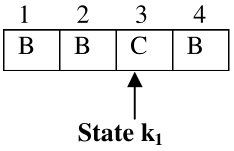
	s_1	s_2	s_3	S_4
s_1	C, k_1	own		
s_2		C	own	
s_3			C	own
s_4				B, end

Applying the transition function δ (k_1,C) = (k_2,B,R) we get:



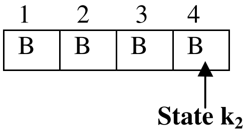
	s_1	s_2	s_3	s_4
s_1	B	own		
s_2		C, k_2	own	
s_3			C	own
s_4				B, end

Applying the transition function δ (k_2,C) = (k_1,B,R) and we get :



	s_1	s_2	s_3	s_4
s_1	B	own		
s_2		B	own	
s_3			C, k_1	own
s_4				B, end

Applying the transition function δ (k_1,C) = (k_2,B,R) and we get :



	s_1	s_2	s_3	s_4
s_1	B	own		
s_2		B	own	
s_3			B	own
s_4				B, k_2, end

This is the last state, because no transition function is applicable now; halting state k_3 is not reached.