3.3 Derive equivalence classes for the input variables listed below.

a. `int` *pen_inventory*; Current inventory level of writing pens.
b. `string` *planet_name*; Planet name.
c. *operating_system*={"OS X", "Windows XP", "Windows 2000", "Unix", "Linux", "Xinu", "VxWorks"}; Name of an operating system.
d. `printer_class=set` printer_name;
`printer_class` p; Set of printer names.
e. `int` *name* [1.10]; An array of at most 10 integers.

3.4 In Example 3.4, suppose now that we add another category of printers, say, "Home and home office (hb)." Define a suitable relation *hb* that partitions the input domain of `pTest` into two equivalence classes. Discuss the overlap of the equivalence classes induced by *hb* with the remaining eight classes defined by the four relations in Example 3.4.

3.5 Consider the following relation

$cl : \mathcal{I} \rightarrow \{yes, no\}$

cl maps an input domain $\mathcal{I}$ of *pTest* in Example 3.4 to the set $\{yes, no\}$. A printer make and model is mapped to *yes* if it is a color laserjet, else it is mapped to *no*. Is *cl* an equivalence relation?

3.6 (a) Why consider classes E2–E6 in Example 3.5 when the correctness of the program corresponding to tests in these classes can be verified by simple inspection? Offer at least two reasons. (b) Are there any additional equivalence classes that one ought to consider while partitioning the input domain of `wordCount`?

3.7 Partition the input domain of the *transcript* component described in Example 3.6 into equivalence classes using the guidelines in Tables 3.1 and 3.2. Note that *transcript* takes two inputs, a record R and an integer N.

3.8 (a) Generate two sets of tests T_1 and T_2 from the partitions created in Example 3.7 using, respectively, uni-dimensional and multidimensional partitioning. Which of the following relations holds amongst T_1 and T_2 that you have created: $T_1 = T_2$, $T_1 \subset T_2$, $T_1 \subseteq T_2$, $T_1 \supset T_2$, $T_1 \supseteq T_2$, and $T_1 \neq T2$? (b) Which of the six relations mentioned could hold between T_1 and T_2 assuming that T_1 is derived from equivalence classes constructed using uni-dimensional partitioning and T_2 using multidimensional partitioning?

3.9 Consider an application *App* that takes two inputs *name* and *age* where *name* is a non-empty string containing at most 20 alphabetic characters and *age* is an integer that must satisfy the constraint $0 \leq age \leq 120$. *App* is required to display an error message if the input value provided for *age* is out of range. It truncates any name that is more than 20 characters in length and generates an error message if an empty string is supplied for *name*.

Partition the input domain using (a) uni-dimensional partitioning and (b) multidimensional partitioning. Construct two sets of test data for *App* using the equivalence classes derived in (a) and in (b).

3.10 Suppose that an application has m input variables and that each variable partitions the input space into n equivalence classes. The multidimensional partitioning approach will divide the input domain into how many equivalence classes?

3.11 An application takes two inputs x and y where $x \leq y$ and $-5 \leq y \leq 4$. (a) Partition the input domain using uni-dimensional and multidimensional partitioning. (b) Derive test sets based on the partitions created in (a).

3.12 In Example 3.8, we started out by calculating the number of equivalence classes to be 120. We did so because we did not account for the parent–child relationship between *cmd* and *tempch*. Given this relationship, how many equivalence classes should we start out with in the first step of the procedure for partitioning the input domain into equivalence classes?

3.13 (a) Identify weaknesses, as many as you can, of the test T derived in Example 3.10.

(b) Derive a test set that covers each individual equivalence class derived for the four variables in Example 3.8 while ensuring that the semantic relations between different variables are maintained.

(c) Compare the test set derived in (b) with that in Table 3.3 in terms of their respective sizes and error detection effectiveness. If you believe that the error detection effectiveness of the test set you derived is less than that of the test set in Table 3.3, then offer an example of an error in the boiler control software that will likely be not detected by your test set but will likely be detected by the test set in Table 3.3.

3.14 An object named *compute* takes an integer x as input. It is required to send a message to another object O_1 if $x \leq 0$ and a message to object O_2 if $x > 0$. However, due to an error in *compute*, a message is sent to to O_1 when $x < 0$ and to O_2 otherwise. Under what condition(s) will the input $x = 0$ *not* reveal the error in *compute*?

3.15 For each test $t \in T$ in Example 3.12, construct one example of a fault in *textSearch* that is guaranteed to be detected only by t. *Hint:* Avoid trivial examples!

3.16 A method named *cC* takes three inputs: *from*, *to*, and *amount*. Both *from* and *to* are strings and denote the name of a country. Variable *amount* is of type `float`. Method *cC* converts *amount* in the currency of the country specified by *from* and returns the equivalent amount, a quantity of type `float`, in the currency of the country specified by *to*. Here is an example prepared on July 26, 2004:

Inputs: *from* = "USA", *to* = "Japan", *amount* = 100
Returned value: 11,012.0

(a) Derive a set of tests for *cC* using equivalence partitioning and boundary values analysis. (b) Suppose that a GUI encapsulates *cC* and allows the user to select the values of *from* and *to* using a palette of country names. The user types in the amount to be converted in a text box before clicking on the button labeled `Convert`. Will the presence of the GUI change the tests you derived in (a)? If so, how? If not, why?

You may find that the requirement specification given for cC is incomplete in several respects. While generating tests, it is recommended that you resolve any ambiguities in the requirements and complete the information not available by using common sense and/or discussing the problem with appropriate members of the design/development team.

3.17 Recall the boundary value analysis in Example 3.11. (a) Construct one example of a boundary error in *fP* that may go undetected unless tests t_2 and t_5 are replaced so that computation at the boundaries of *code* and *qty* is checked in separate tests. (b) Replace t_2 and t_5 by an appropriate set of tests that test *fP* at the boundaries of *code* and *qty* in separate tests.

3.18 Compare test generation using boundary value analysis and equivalence partitioning methods described in this chapter in terms of the number of tests generated, the error detection ability, and the source of tests.

3.19 Consider the following problem for computing the monthly utility bill as formulated by Rob Hierons to illustrate coincidental correctness.

Let W and E denote, respectively, the monthly water and electricity consumption in standard units. Let C_w and C_e denote, respectively, the costs of consuming one standard unit of water and electricity. The monthly charge to the customer is computed as: $(C_w * W + C_e * E)$. However, in situations where a customer uses at least M_w units of water, a 20% discount is applied to the electricity portion of the charge. In this case, the charge to the customer is computed as $(C_w * W + 0.2 * C_e * E)$.

Now suppose that $M_w = 30$ but while coding the computation of the monthly utility charge, the programmer has used by mistake $M_w = 40$. Thus, instead of using the condition $W \geq 30$ the programmer uses the condition $W \geq 40$ to select one of the two formulae above to compute the monthly utility charges. Assume that the function under test (in Java) is as given below.

```
public double incorrectHierons(double w, double e, double cw,
double ce){
  final double mw=40; // Should be 30.
  final double discount=0.2;
  if(w>=mw){
    return(w*cw+discount*ce*e);
  }else{
  return(w*cw+ce*e);
  }
}
```

(a) Use boundary value analysis to determine test inputs to test the (incorrect) function to compute the utility bill. (b) Use partition testing techniques to derive the test cases to test the (incorrect) function to compute the utility bill. (c) Under what conditions will your test fail to discover the error?