The University of Melbourne

SWEN90006: Software Testing and Reliability

Tutorial 2 Solutions

Semester 2, 2018

Introduction

The aim of this tutorial is twofold. First, it aims to give you some practise at deriving test cases from specifications. Second, it aims for you to start exploring the limits of your test cases, and of the specifications.

The Program

Introduction: LWIG (Less Work Is Good) is a new marking program developed for the department of Information Programming.

The LWIG program takes a .in file as an argument and produces a sorted .out file. The program assumes that the .in file contains at least one student record.

Input File: The input file format is as follows. Each line will contain the data for a single student, and will contain several fields. Each field is separated by a colon.

Each line consists of the following fields, in order:

- A student number, which must be a 5 digit, 6 digit or 9 digit number.
- The student's month of birth, which must be a string of 3 alphabetic characters with the first character capitalised, for example, "Mar", or "MAR".
- The date of birth, which must be a number from 1–31 and must be the correct number of days for the month; that is, it cannot be 30 February, because February never has 30 days.
- The student's surname, which must be a string of alphabetic characters all in capitals.
- The first letter of the student's first name, which must be a single capitalised alphabetic character.
- The number of lectures that they slept through, which must be an integer between 0 and 24.

If any input row is invalid, the program should print a warning message and continue with the next record. If the program encounters a more serious problem (e.g. unable to open input file), it will print an error message and exit gracefully.

Example 1 As an example suppose we had the following data.

- Student number: 12345
- Month of Birth: May
- Date of Birth: 26

- Surname: CHAN
- First letter of first name: K
- Slept during one or more lectures: 0

The input line for this data would be `12345:MAY:26:CHAN:K:0`

Output File: The output file format is as follows. Each line will contain the data for a single student, and each line will contain several fields. Each field is separated by a colon.

Each line consists of the following fields, in order:

- Student number which must be a 5 digit, 6 digit, or 9 digit number.
- Assignment 1 mark: (0–10)
- Assignment 2 mark: (0–10)
- Project mark: (0–30)
- Exam mark: (0-50)
- Final grade: [H1, H2A, H2B, H3, P, HCP, N, N+]
- Comment: String of arbitrary length

Example 2 As an example, consider the following output:

- Student number: 12345
- Assignment 1 mark: 10
- Assignment 2 mark: 10
- Project mark: 25
- Exam mark: 38
- Final grade: H1
- Comment: "Excellent."

The output for this record would be `12345:10:10:25:38:H1:‘‘Excellent’’`

Tasks

Question 1

What is the input domain for the LWIG program? What are the input conditions for the LWIG program?

Answer

The trick here is to think in terms of the logical input variables to the program and then to break this down further into the fields. The input is a file consisting of the following logical *field* variables. The fields and their input domains and input conditions are as follows.

(i) Student number — The student number is just a string, although the intent is that it is a string of digits. Any string of 5 characters, 6 characters or 9 characters may appear in this field position and so the input domain is the set of strings.

The input condition here is that the student number must be a string consisting of 5 digits, 6 digits or 9 digits. We can formalise this if we wish by saying that our logical field variable student_no must satisfy the input condition:

$$\mathsf{student\_no} \in \{S \mid \mathsf{length[S]} = 5 \wedge \text{for each i in the domain of S, } \mathsf{S[i]} \in \mathsf{digits}\}$$

OR

$$\mathsf{student\_no} \in \{S \mid \mathsf{length[S]} = 6 \wedge \mathsf{for\ each\ i\ in\ the\ domain\ of\ S,\ \ S[i]} \in \mathsf{digits}\}$$

OR

$$\mathsf{student\_no} \in \{S \mid \mathsf{length[S]} = 9 \wedge \mathsf{for\ each\ i\ in\ the\ domain\ of\ x,\ \ S[i]} \in \mathsf{digits}\}$$

where digits is the set of characters 0–9.

(ii) Month of birth and day of birth — It does not make much sense to treat months and days in isolation *because of the dependency between days and months*.

Unfortunately, building up this sort of judgement is part of building up the experience in testing. Since the logical variables here are three letter strings and integers in a range the input domain here can be considered to be pairs of (Strings, Integers).

The input condition is that for any pair (Month, Date) we must have

$$\begin{array}{lll} (\mathsf{Month} = \text{"Jan"} \ \ \mathsf{or}\ \ \mathsf{Month} = \text{"JAN"}) & \mathsf{and} & \mathsf{Date} \in 1..31 \\ (\mathsf{Month} = \text{"Feb"} \ \ \mathsf{or}\ \mathsf{Month} = \text{"FEB"}) & \mathsf{and} & \mathsf{Date} \in 1..29 \\ (\mathsf{Month} = \text{"Mar"} \ \ \mathsf{or}\ \mathsf{Month} = \text{"MAR"}) & \mathsf{and} & \mathsf{Date} \in 1..31 \\ \vdots & & \end{array}$$

(iii) Family name — The input domain for this field is the set String and the input condition is that all elements of the string must be capital letters; that is, given a family name S, then for each $0 \leq \mathsf{i} \leq \mathsf{length[S]} - 1$, we have $\mathsf{S[i]} \in$ A–Z.

(iv) Initial — Likewise, the input domain for the student's initial is the set A–Z of characters.

(v) No. of lectures during which the student slept — The input domain is the set of integers and the input condition is that the number of lectures is between 0 and 24; that is, $0 \leq \mathsf{slept} \leq 24$.

(vi) Finally, we must not forget the input file. The input domain is the set of all files that can be read. The input condition is that there is at least one record in the file, and that for every record in the file, every field in that record conforms to the input conditions above.

Question 2

Derive input test-cases for the program using equivalence partitioning and boundary-value analysis.

Answer

First, we examine each requirement, which relate to specific variables:

(i) Student number — Applying to guideline 2, there are four equivalence classes: strings of 5 digit numbers, strings of 6 digit numbers and strings of 9 digit numbers, and anything else.

The **valid** classes are the sets of 5 digit, 6 digit and 9 digit strings:

$$\begin{aligned} \mathsf{EC}_5 &= \{\mathsf{S} \mid \mathsf{length[S]} = 5 \wedge \mathsf{for\ each\ i\ in\ the\ domain\ of\ S,\ \ S[i]} \in \mathsf{digits}\} \\ \mathsf{EC}_6 &= \{\mathsf{S} \mid \mathsf{length[S]} = 6 \wedge \mathsf{for\ each\ i\ in\ the\ domain\ of\ S,\ \ S[i]} \in \mathsf{digits}\} \\ \mathsf{EC}_9 &= \{\mathsf{S} \mid \mathsf{length[S]} = 9 \wedge \mathsf{for\ each\ i\ in\ the\ domain\ of\ S,\ \ S[i]} \in \mathsf{digits}\} \end{aligned}$$

where digits is the set of characters 0–9.

The **invalid** classes are all the other strings.

(ii) Months and day of birth — Use guideline 2 for months. In theory, we should select 24 different cases: one for each month capitalised (e.g. "JAN") and un-capitalised (e.g. "Jan"). However, this may be overkill. To keep it simple, we choose one month with 31 days (Jan), one with 30 days (Nov), and one with 28 days (Feb), and an invalid case of something other than a month, but choosing all 12 months would also be fine. In the tree below, we omit JAN, FEB, and NOV to simplify, but we should als

Then, apply guideline 1 to the possible dates of each. It is important that we test the number of dates for Jan, Feb, and Nov, because the behaviour is different for months of different lengths[1]:

$$
\begin{array}{lcl}
\mathsf{EC}_{\mathsf{ValidJan}} & = & \{(\mathsf{Month,Date}) \mid \ \mathsf{Month} = \text{“}\mathsf{Jan}\text{”}) \text{ and Date} \in 1..31\} \\
\mathsf{EC}_{\mathsf{InValidJan\_1}} & = & \{(\mathsf{Month,Date}) \mid \ \mathsf{Month} = \text{“}\mathsf{Jan}\text{”}) \text{ and Date} < 1\} \\
\mathsf{EC}_{\mathsf{InValidJan\_2}} & = & \{(\mathsf{Month,Date}) \mid \ \mathsf{Month} = \text{“}\mathsf{Jan}\text{”}) \text{ and Date} > 31\} \\
\ldots & & \text{Similar for Feb and Nov} \\
\mathsf{EC}_{\mathsf{Invalid\ M}} & = & \{\ (\mathsf{M,\ D}) \mid \mathsf{M} \notin \{\text{“Jan”, “JAN”, “Feb”, “FEB”}, \ldots\}\}
\end{array}
$$

Applying this to the leaf nodes of the length of the student number, the assumption is that the length of the student number is independent of the month, so we apply it just to one of those leaf nodes: length of 6. This choice is arbitrary.

(iii) Family name — The input condition gives a set of valid inputs so use guideline 2 again to get:

$$
\begin{array}{lcl}
\mathsf{EC}_{\mathsf{Valid\ FN}} & = & \{\ \mathsf{S} \mid \mathsf{S[i]} \in \text{A–Z for some } 0 \le \mathsf{i} \le \text{length[S] - 1}\} \\
\mathsf{EC}_{\mathsf{Invalid\ FN}} & = & \{\ \mathsf{S} \mid \mathsf{S[i]} \notin \text{A–Z for some } 0 \le \mathsf{i} \le \text{length[S] - 1}\}
\end{array}
$$

Again, we feel that this is independent of dates or student numbers, so we aribtrarily pick a valid leaf node to apply it to: Nov.

(iv) The Initial – Using guideline 2, we get a valid EC $\mathsf{EC}_{\mathsf{Valid\ I}}$ of all capital letters and an invalid EC $\mathsf{EC}_{\mathsf{Invalid\ I}}$ of all characters except those in $\mathsf{EC}_{\mathsf{Valid\ I}}$.

(v) The no. of lectures slept through — The input condition gives a range of values and so use guideline 1. This gives one valid EC and two invalid ECs as follows:

$$
\begin{array}{lcl}
\mathsf{EC}_{\mathsf{Valid\ Sleeps}} & = & 0\,..\,24 \\
\mathsf{EC}_{\mathsf{Invalid\ Sleeps_1}} & = & \{\mathsf{S} \mid \mathsf{S} < 0\} \\
\mathsf{EC}_{\mathsf{Invalid\ Sleeps_1}} & = & \{\mathsf{S} \mid \mathsf{S} > 24\}
\end{array}
$$

(vi) Additionally, we should test that the program can handle cases of 0, 1, and many records in a file. These could be included with the existing test cases above, but we omit for brevity.

The resulting test template tree is shown in the appendix.

Selecting test cases

Using boundary-value analysis, we select test cases for the months including the first and last day of the month, and invalid cases either side; similarly for the number of days slept.

For Jan, we have:

Date = 1	(on point)
Date = 31	(on point)
Date = 0	(off point)
Date = 32	(off point)

For the surname, we select zero characters outside of A-Z as the on point, and one character outside of A-Z as the off point.

For the lengths, we need to select lengths of 5, 6 and 9 as on-points, and 4, 7, 8, and 10 as off points. Note that the off points of 6 for length[S] = 5 and 5 for length[S] = 6 are already tested as on points for each other.

[1]Note, we ignore the leap year/non-leap year distinction here.

For EC13, the ‘for some i in 1..S#’, we can have 0 or > 0 such occurences of lower-case letters, so the on point is 0 (which falls into EC12) and the off point is 1 lower-case letter.

For c in $A - Z$, we use the ASCII chart to find the off point of `@` (the character immediately below A) and `[` (the character immediately above Z).

The resulting tests are:

EC1	`12345:Jan:1:ABC:A:0`	on point $length[S] = 5$
	`1234:Jan:1:ABC:A:0`	off point below $length[S] = 5$
EC2	`123456:Jan:0:ABC:A:1`	off point for $Date < 1$
EC3	`123456:Jan:1:ABC:A:-1`	off point for $Sleeps < 0$
EC4	`123456:Jan:1:ABC:A:0`	on point for $Sleeps < 0$ and $Date < 1$
	`123456:Jan:31:ABC:A:25`	on point for $Sleeps > 24$ and $Date > 31$
EC5	`123456:Jan:31:ABC:A:24`	off point for $Sleeps > 24$
EC6	`123456:Jan:32:ABC:A:1`	off point for $Date > 31$
EC7	`123456:Feb:0:ABC:A:1`	off point for $Date < 1$
EC8	`123456:Feb:1:ABC:A:1`	on point for $Date < 1$
	`123456:Feb:29:ABC:A:1`	on point for $Date > 29$
EC9	`123456:Feb:30:ABC:A:1`	off point for $Date > 29$
EC10	`123456:Nov:0:ABC:A:1`	off point for $Date < 1$
EC11	`123456:Nov:1:ABC:A:1`	on point for $Date < 1$ and $c < A$
	`123456:Nov:30:ABC:Z:1`	on point for $Date > 30$ and $c > A$
EC12	`123456:Nov:1:ABC:@:1`	on point for $Date < 1$ and off point for $c < A$
	`123456:Nov:30:ABC:[:1`	on point for $Date > 30$ and off point $c > A$
EC13	`123456:Nov:30:aBC:[:1`	on point for ‘for some i’
EC14	`123456:Nov:31:ABC:Z:1`	off point for $Date > 30$
EC15	`123456:Nov:30:ABC:Z:1`	off point for $Date > 30$
EC16	`123456789:Nov:30:ABC:Z:1`	on point for $length[S] = 9$
EC17	`1234567:Nov:30:ABC:Z:1`	off point for $length[S] = 6$
	`12345678:Nov:30:ABC:Z:1`	off point for $length[S] = 9$
	`1234567890:Nov:30:ABC:Z:1`	off point for $length[S] = 9$

Note that some equivalence classes have more than one test because there are multiple boundaries. For example, $Date$ in 1..31 has the boundary $Date \geq 1$ and the boundary $Date \leq 31$.

Question 3

Of course, the client has not completely specified the program (but, that is to be expected). There is an additional requirement that the records can be sorted by different output fields.

What are the implications of sorting on the various fields and what test cases would you choose to ensure that sorting has been correctly implemented?

Answer

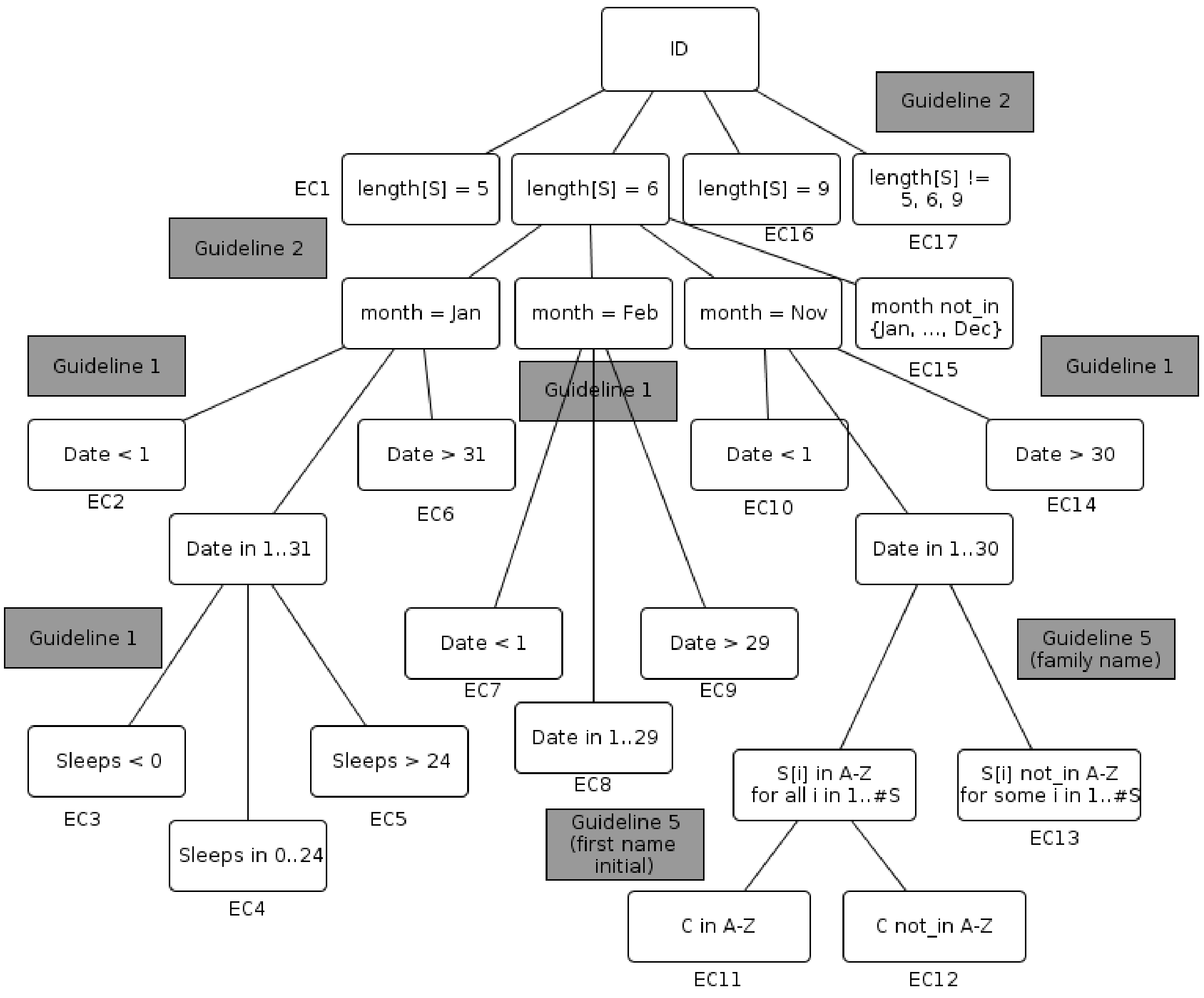
The implication is that we would have to consider the order of the records in the file. This could be tested using a separate set of test cases, or the existing test cases could be re-arranged to consider sorting.

At the very least, we would want to test the following:

- A file containing 1 record;
- Two files containing 2 records – one sorted and one not; and
- Three files contains more than 2 records – one sorted, one reverse-sorted, and one completed unsorted.

We would need to do this for every field that is common between the input and output; for example, the student number. To test these fields, we could use the existing test cases and sort them on their output. That is, for the final grade, use the existing test cases, sort them based on final grade, and check that the final grade column is sorted.

A Test template tree for the LWIG program



ID
Guideline 2
EC1
length[S] = 5
length[S] = 6
length[S] = 9
length[S] != 5, 6, 9
EC16
EC17
Guideline 2
month = Jan
month = Feb
month = Nov
month not_in {Jan, ..., Dec}
EC15
Guideline 1
Guideline 1
Guideline 1
Date < 1
EC2
Date > 31
EC6
Date < 1
EC10
Date > 30
EC14
Date in 1..31
Date in 1..30
Guideline 1
Date < 1
EC7
Date > 29
EC9
Guideline 5 (family name)
Date in 1..29
EC8
Sleeps < 0
EC3
Sleeps > 24
EC5
Sleeps in 0..24
EC4
S[i] in A-Z for all i in 1..#S
S[i] not_in A-Z for some i in 1..#S
EC13
Guideline 5 (first name initial)
C in A-Z
EC11
C not_in A-Z
EC12