uc3m | Universidad Carlos III de Madrid

TEST CASES DESCRIPTION

Guided Exercise 2
Software Development

G87.2024.T3.GE2

Function 1 - roomReservation

In this function, we used the Equivalence Classes and Boundary Values approach to testing. The function receives 7 inputs for which we had to analyze in terms of the values that they are able to take: that is, their datatype, their format, their range of values… We designed 6 of our variables as strings (credit_card, name_surname, id_card, phone_number, room_type, arrival_date) as 1 of them as an int (num_days). We also had 2 types of outputs: a valid MD5 string that served as a localizer, and exceptions of different types we had to define (for example, depending on which of the variables raised the exception).

We ended up with 28 valid Equivalence Classes, 20 valid Boundary Values, 68 non-valid Equivalence Classes and 26 non-valid Boundary Values.

Here you can find the definition of each of them, by variable and criteria:

VARIABLE	CRITERIA	VALID CLASSES	INVALID CLASSSES
credit_card	Datatype	ECV1 string with valid format ("5105105105105100")	ECNV1 int (5105105105105100)
	Length: 16 digits	ECV2 16 digits ("5105105105105100")	ECNV2 less than 16 digits (15) ("510510510510")
		BVV1 16 digits ("5105105105105100")	ECNV3 more than 16 digits (17) ("51051051051051000")
			BVNV1 15 digits ("510510510510")
			BVNV2 17 digits ("51051051051051000")
	Valid 16 number with Luhn algorithm	ECV3 valid credit card ("5105105105105100")	ECNV4 we have 16 characters but not all of them are digits ("A105105105105100")
			ECNV5 credit card number with 16 digits but the last one is wrong (does not follow the Luhn algorithm) ("5105105105105101")
name_surname	Datatype	ECV4 string with valid format ("John Smith")	ECNV6 int (1)
	At least 2 strings	ECV5 2 or more strings separated by 1 white space ("John Smith")	ECNV7 less than 2 strings (1) ("JohnSmithy")
		BVV2 2 strings ("John Smith")	BVNV3 1 string ("JohnSmithy")
		BVV3 3 strings ("John Jack Smith")	
	Strings separated by 1 white space	ECV6 2 strings separated by 1 white space ("John Smith")	ECNV8 less than 1 white space (0) ("JohnSmithy")

		BVV4 1 white space ("John Smith")	ECNV9 more than 1 white space (2) ("John Smith")
			BVNV4 0 white spaces ("JohnSmithy")
			BVNV5 2 white spaces ("John Smith")
	Length: between 10 and 50 characters	ECV7 between 10 and 50 characters (10) ("John Smith")	ECNV10 less than 10 characters (9) ("John Smit")
		BVV5 10 characters ("John Smith")	ECNV11 more than 50 characters (51) ("John SmithSmithSmithSmithSmithSmithSmithSmithSmithy")
		BVV6 11 characters ("John Smithy")	BVNV6 9 characters ("John Smit")
		BVV7 49 characters ("John SmithSmithSmithSmithSmithSmithSmithSmithSmit")	BVNV7 51 characters ("John SmithSmithSmithSmithSmithSmithSmithSmithSmithy")
		BVV8 50 characters ("John SmithSmithSmithSmithSmithSmithSmithSmithSmith")	
id_card	Datatype	ECV8 string with valid format ("12345678Z")	ECNV12 int (12345678)
	8 digits at the beginning	ECV9 8 digits at the beginning ("12345678Z")	ECNV13 we have 8 characters at the beginning but not all of them are digits ("AB12C678Z")
			ECNV14 less than 8 digits (7) ("1234567Z")
			ECNV15 more than 8 digits (9) ("123456789Z")
			BVNV8 7 digits ("1234567Z")
			BVNV9 9 digits ("123456789Z")
	1 letter at the end	ECV10 1 letter at the end ("12345678Z")	ECNV16 the last character is not a letter nor a digit ("12345678#")
			ECNV17 less than 1 letter at the end (0) ("12345678")
			ECNV18 more than 1 letter at the end (2) ("12345678ZZ")
			BVNV10 0 letters at the end ("12345678")
			BVNV11 2 letters at the end ("12345678ZZ")
	Valid letter algorithm	ECV11 valid letter ("12345678Z")	ECNV19 invalid letter ("12345678A")
phone_number	Datatype	ECV12 string with valid format ("612345789")	ECNV20 int (612345789)

	9 digits	ECV13 string with 9 digits ("612345789")	ECNV21 string with 9 characters but not all of them are digits ("A12345789")
		BVV9 9 digits ("612345789")	ECNV22 string with less than 9 digits (8) ("61234578")
			ECNV23 string with more than 9 digits (10) ("6123457890")
			BVNV12 8 digits ("61234578")
			BVNV13 10 digits ("6123457890")
room_type	Datatype	ECV14 string with valid value ("single")	ECNV24 int (1)
	Allowed values	ECV15 "single"	ECNV25 any other value ("other")
		ECV16 "double"	
		ECV17 "suite"	
arrival_date	Datatype	ECV18 string with valid format ("01/07/2024")	ECNV26 int (172024)
	Day is 2 digits	ECV19 day is 2 digits ("01/07/2024")	ECNV27 day is 2 characters but not all of them are digits ("A1/07/2024")
		BVV10 day is 2 digits ("01/07/2024")	ECNV28 day is less than 2 digits (1) ("1/07/2024")
			ECNV29 day is more than 2 digits (3) ("001/07/2024")
			BVNV14 day is 1 digit ("1/07/2024")
			BVNV15 day is 3 digits ("001/07/2024")
	Month is 2 digits	ECV20 month is 2 digits ("01/07/2024")	ECNV30 month is 2 characters but not all of them are digits ("01/A7/2024")
		BVV11 month is 2 digits ("01/07/2024")	ECNV31 month is less than 2 digits (1) ("01/7/2024")
			ECNV32 month is more than 2 digits (3) ("01/007/2024")
			BVNV16 month is 1 digit ("01/7/2024")
			BVNV17 month is 3 digits ("01/007/2024")
	Year is 4 digits	ECV21 year is 4 digits ("01/07/2024")	ECNV33 year is 4 characters but not all of them are digits ("01/07/A024")
		BVV12 year is 4 digits ("01/07/2024")	ECNV34 year is less than 4 digits (3) ("01/07/024")
			ECNV35 year is more than 4 digits (5) ("01/07/02024")
			BVNV18 year is 3 digit ("01/07/024")
			BVNV19 year is 5 digits ("01/07/02024")
	"/" between day and month	ECV22 "/" between day and month ("01/07/2024")	ECNV36 other character between day and month ("01-07/2024")

		BVV13 1 "/" between day and month ("01/07/2024")	ECNV37 less than 1 "/" between day and month ("0107/2024")
			ECNV38 more than 1 "/" between day and month ("01//07/2024")
			BVNV20 0 "/" between day and month ("0107/2024")
			BVNV21 2 "/" between day and month ("01//07/2024")
	"/" between month and year	ECV23 "/" between month and year ("01/07/2024")	ECNV39 other character between month and year ("01/07-2024")
		BVV14 1 "/" between month and year ("01/07/2024")	ECNV40 less than 1 "/" between month and year ("01/072024")
			ECNV41 more than 1 "/" between month and year ("01/07//2024")
			BVNV22 0 "/" between month and year ("01/072024")
			BVNV23 2 "/" between month and year ("01/07//2024")
	Valid date algorithm	ECV24 date that exists ("01/07/2024")	ECNV42 date that does not exist ("91/07/2024")
		ECV25 date is the defined current date or after it ("01/07/2024")	ECNV43 date before the defined current date ("01/02/2024")
		BVV15 date is the defined current date ("01/07/2024")	BVNV24 date is a day before the defined current date ("30/06/2024")
		BVV16 date is a day after the defined current date ("02/07/2024")	
num_days	Datatype	ECV26 int in valid range (1)	ECNV44 string ("1")
	1 <= num_days <= 10	ECV27 1 <= num_days <= 10	ECNV45 num_days < 1 (0)
		BVV17 1	ECNV46 num_days > 10 (11)
		BVV18 2	BVNV25 0
		BVV19 9	BVNV26 11
		BVV20 10	
--	Outputs	ECV28 valid MD5 string	ECNV47 Exception: credit_card is not a string
			ECNV48 Exception: credit_card is not 16 characters long

			ECNV49 Exception: credit_card must have 16 digits
			ECNV50 Exception: credit_card does not follow the Luhn algorithm
			ECNV51 Exception: name_surname is not a string
			ECNV52 Exception: name_surname must contain at least 2 strings separated by a white space
			ECNV53 Exception: name_surname must be between 10 and 50 characters long
			ECNV54 Exception: id_card is not a string
			ECNV55 Exception: id_card must have 8 digits and 1 final letter
			ECNV56 Exception: invalid letter for id_card
			ECNV57 Exception: a client with specified id_card already has a reservation
			ECNV58 Exception: phone_number is not a string
			ECNV59 Exception: phone_number is not 9 characters long
			ECNV60 Exception: phone_number must have 9 digits
			ECNV61 Exception: room_type is not a string
			ECNV62 Exception: invalid room_type value
			ECNV63 Exception: arrival_date is not a string
			ECNV64 Exception: invalid arrival_date format "DD/MM/YYYY"
			ECNV65 Exception: arrival_date does not exist
			ECNV66 Exception: arrival_date before current date
			ECNV67 Exception: num_days is not an int
			ECNV68 Exception: num_days must be between 1 and 10

Once we had this, we moved on to describe valid and invalid test cases that would cover all of the Equivalence Classes and Boundary Values. In the valid test cases, we were sometimes able to cover more than one valid Equivalence Class, but for the invalid test cases, we made sure we only tested one invalid Equivalence Class, along with its corresponding exception Equivalence Class. That is to say, we only tested one invalid input at a time, because mixing up multiple invalid inputs makes it hard to know where the bugs of the code may come from.

We ended up with 57 total test cases. Here is the breakdown by variable:

- credit_card: 6 test cases (1 valid + 5 invalid)
- name_surname: 9 test cases (4 valid + 5 invalid)
- id_card: 9 test cases (9 invalid)
- phone_number: 4 test cases (4 invalid)
- room_type: 4 test cases (2 valid + 2 invalid)
- arrival_date: 19 test cases (1 valid + 18 invalid)
- num_days: 6 test cases (3 valid + 3 invalid)

Here is the description of each test. Their complete definition can be found in the excel file:

FIELD	VALID/INVALID	ID TEST	DESCRIPTION
credit_card	VALID	TC1	Valid credit_card
credit_card	INVALID	TC2	credit_card of datatype int
credit_card	INVALID	TC3	credit_card has 15 digits
credit_card	INVALID	TC4	credit_card has 17 digits
credit_card	INVALID	TC5	credit_card has 16 characters but not all of them are digits
credit_card	INVALID	TC6	credit_card does not follow Luhn algorithm (last digit is wrong)
name_surname	VALID	TC7	name_surname with more than 2 strings separated by a white space
name_surname	VALID	TC8	name_surname has 11 characters
name_surname	VALID	TC9	name_surname has 49 characters
name_surname	VALID	TC10	name_surname has 50 characters
name_surname	INVALID	TC11	name_surname of datatype int
name_surname	INVALID	TC12	name_surname has only 1 string (0 white spaces)
name_surname	INVALID	TC13	name_surname has more than 1 white space between 2 strings
name_surname	INVALID	TC14	name_surname has 9 characters
name_surname	INVALID	TC15	name_surname has 51 characters
id_card	INVALID	TC16	id_card of datatype int
id_card	INVALID	TC17	id_card has 8 characters at the beginning but not all of them are digits
id_card	INVALID	TC18	id_card has less than 8 digits
id_card	INVALID	TC19	id_card has more than 8 digits
id_card	INVALID	TC20	The last character of id_card is not a letter nor a digit
id_card	INVALID	TC21	id_card has less than 1 letter at the end
id_card	INVALID	TC22	id_card has more than 1 letter at the end

id_card	INVALID	TC23	Invalid letter for id_card according to the algorithm
id_card	INVALID	TC24	id_card belongs to a client that already has a reservation file
phone_number	INVALID	TC25	phone_number of datatype int
phone_number	INVALID	TC26	phone_number has 9 characters but not all of them are digits
phone_number	INVALID	TC27	phone_number has less than 9 digits
phone_number	INVALID	TC28	phone_number has more than 9 digits
room_type	VALID	TC29	Valid room_type value ("double")
room_type	VALID	TC30	Valid room_type value ("suite")
room_type	INVALID	TC31	room_type of datatype int
room_type	INVALID	TC32	Invalid room_type value
arrival_date	VALID	TC33	arrival_date is 1 day after the current time
arrival_date	INVALID	TC34	arrival_date of datatype int
arrival_date	INVALID	TC35	Day of arrival_date is 2 characters but not all of them are digits
arrival_date	INVALID	TC36	Day of arrival_date is less than 2 digits
arrival_date	INVALID	TC37	Day of arrival_date is more than 2 digits
arrival_date	INVALID	TC38	Month of arrival_date is 2 characters but not all of them are digits
arrival_date	INVALID	TC39	Month of arrival_date is less than 2 digits
arrival_date	INVALID	TC40	Month of arrival_date is more than 2 digits
arrival_date	INVALID	TC41	Year of arrival_date is 4 characters but not all of them are digits
arrival_date	INVALID	TC42	Year of arrival_date is less than 4 digits
arrival_date	INVALID	TC43	Year of arrival_date is more than 4 digits
arrival_date	INVALID	TC44	arrival_date has a character between day and month that is not "/"
arrival_date	INVALID	TC45	arrival_date has less than 1 "/" between day and month
arrival_date	INVALID	TC46	arrival_date has more than 1 "/" between day and month
arrival_date	INVALID	TC47	arrival_date has a character between month and year that is not "/"
arrival_date	INVALID	TC48	arrival_date has less than 1 "/" between month and year
arrival_date	INVALID	TC49	arrival_date has more than 1 "/" between month and year
arrival_date	INVALID	TC50	arrival_date does not exist

arrival_date	INVALID	TC51	arrival_date is before the current time
num_days	VALID	TC52	num_days is 2
num_days	VALID	TC53	num_days is 9
num_days	VALID	TC54	num_days is 10
num_days	INVALID	TC55	num_days of datatype string
num_days	INVALID	TC56	num_days is 0
num_days	INVALID	TC57	num_days is 11

Function 2 - guestArrival

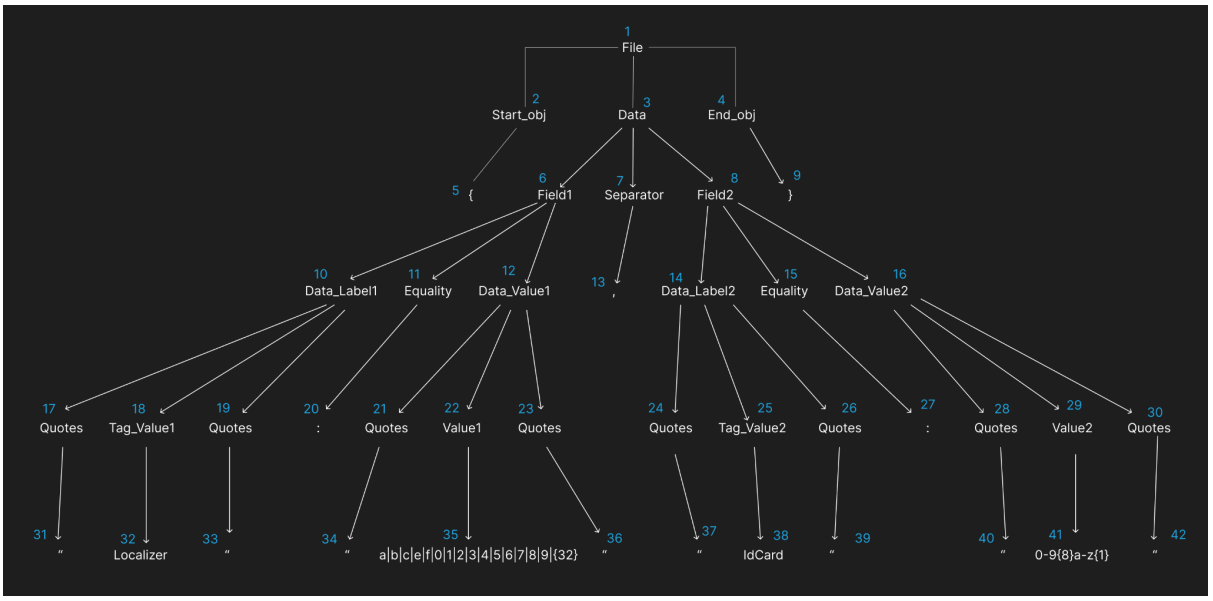
In this function the testing technique that we have applied the most is the Syntax Analysis, as the input of this function is a JSON file. This kind of file stores information following a determined pattern that we have described using a Type 2 grammar:

```
{
    "Localizer":"<String having 32 hexadecimal characters>",
    "IdCard": "<valid idCard>"
}
```

(The format of the input JSON file)

```
File ::= Start_obj Data End_obj
Start_obj ::= {

Data ::= Field1 Separator Field2

Field1 ::= Data_Label1 Equality Data_Value1
Field2 ::= Data_Label2 Equality Data_Value2
Separator ::= ,
Equality ::= :

Data_Label1 ::= Quotes Tag_Value1 Quotes
Tag_Value1 ::= Localizer
Data_Value1 ::= Quotes Value1 Quotes
Value1 ::= a|b|c|d|e|f|0|1|2|3|4|5|6|7|8|9|{32}
Quotes ::= "

Data_Label2 ::= Quotes Tag_Value2 Quotes
Tag_Value2 ::= IdCard
Data_Value2 ::= Quotes Value2 Quotes
Value2 ::= 0-9{8}a-z{1}

End_obj ::= }
```

After defining the grammar that represents the JSON file we need to draw the derivation tree and number the nodes from top to bottom and from left to right:



After this, we are able to identify all the different test cases that we need to do to the function in order to ensure that it works correctly:

- Firstly, we have one test with a valid case, which covers all the nodes of our tree.
- Then, we have 25 tests which verify cases in which one node of the derivation tree has been duplicated.
- Other 25 tests that verify cases in which one node of the derivation tree has been deleted.
- 15 more tests that verify cases in which one terminal node of the derivation tree has been modified.
- Lastly, 7 tests which verify other cases that fall out of the Syntax Analysis we have done for the function.

Function 3 - guestCheckout

Basic Path Descriptions:

Valid Path (Successful Checkout):

Start → Node 1 (valid) → Node 2 (exists) → Node 3 (matches) → Node 4 (success) → End.
This path represents the ideal scenario where every check passes, and the checkout is successfully recorded.

Invalid Room Key Format Path:
Start → Node 1 (invalid) → Terminal Node A.
This path is taken when the room key does not meet the MD5 format requirements.

Room Key File Missing Path:

Start → Node 1 (valid) → Node 2 (missing) → Terminal Node B.
This path occurs if the room key is valid, but the corresponding file does not exist in the storage.

Departure Date Mismatch Path:

Start → Node 1 (valid) → Node 2 (exists) → Node 3 (mismatch) → Terminal Node C.
This path is for scenarios where the room key and file are valid, but the departure date doesn't match today's date.

Record Failure Path:

Start → Node 1 (valid) → Node 2 (exists) → Node 3 (matches) → Node 4 (failure) → Terminal Node D.
This path represents a failure to record the checkout data, despite all other checks passing.