Question 27

For this program, I believe using Special-Value Testing is appropriate. I believe so because Special-Value testing is up to the skill of the tester, but with a program like this (where it’s information such as which month has how man dates and when a year is a leap year) is very common information so finding values to pick would not be a problem. This also will result in no redundancies, and the number of test cases can be anywhere from 1 to the maximum number of possible inputs (but I won’t pick all of them). There are some I would choose that I deem very important.

I would enter a random date in the middle of every month in the same year as an input. I would do this as a basic test case just to see the output. This case would be to see if the program works on this easy level since the possible errors can stem from other dates.

I would enter December 31st of any year as an input. I would do this to see if the program correctly lists January 1st of the following year as the next date. I would do this to test the transition of the program from one year to another.

I would test the last date of every month of a random year (by random, I mean I would just pick a year for every case since this shouldn’t matter for most) to see if the program correctly handles the transition of one month to another, and if it has the correct number of dates listed for all the months. For February, I would enter February 28th on a leap year and non-leap year, and February 29th of a leap year and non-leap year to see if the leap years are programmed correctly. For the non- leap years I would test if the output date listed is correct. I would see if it lists March 1st if I enter February 28th of a non-leap year, and see if it outputs “Invalid Date” if I enter February 29th of a non-leap year. I would then test a valid date in the year 9999, to so if the program can correctly output 10,000 as a year since the maximum digits the user is allowed to enter is four. I would also test a valid date in the year 0000 to see if the program correctly handles that date as well.

I would also test a few years for the leap year dates. I would enter a year that is divisible by four, but not a century year to see if the output is March 1st and February 29th when I enter February 29th and February 28th respectively. I then test a century year that is divisible by four and 400 to see if I get the same outputs. I would then test a century year, such as 100, that is divisible by four and not four hundred. I would do this to see if the program correctly knows that that is not a leap year, and outputs March 1st, and “Invalid Date” for my inputs respectively.

From here I have tested if the programmer has correctly tested the number of months in a year, how many dates each month has, and which year is a leap year. So from here I would get test the expected invalid inputs. I would enter a day that is less than two digits, and another day that is more than 2 digits to test the expected output of that error. I would do the exact same for month as well. For year, I would do very similar and enter a year that was less than four digits, and a year that was more than four. Then I would enter negative numbers for each and see how the program handles those and if the output is “Invalid Date”.

Finally, I would enter strings/characters for the day, month and year each as their own input to test the code outputs the “Incorrect Format” output.

From here, I would have tested all the important possible inputs and would have a very good understanding of how the program works and if anything was wrong with it. For this type of testing, I looked at the requirements and tested dates which didn’t meet the requirements, that easily met the requirements, and which dates were on the border of not meeting or meeting the requirements.