CIS640 Mid term Exam

03/10/2016

Maximum points: 60 + 9

Duration: 75 minutes

Instructions:

A. On the first page of your solution, please write your name, KSU ID, and the total number of pages in your solution.
B. Number each page used in your solution.
C. Make sure you sign the roster both before starting the exam and after completing the exam.
D. The exam is closed resource. So, access to any sort of resources (including mobile phones/devices) is not permitted during the exam. Except pen, pencil, and eraser, all resources should be placed under the table.
E. Use of mobile phones during the exam is not allowed. So, please turn them off.
F. Please return the exam with your solution.

Questions:

1. Define fault, error, and failure. **(3 points)**

   Fault is the mistake (or cause of error).

   Error is the impact of the fault on program states; **Error** is the difference between correct and incorrect states.

   Failure is the event when the program behaves incorrectly.

2. With code example(s), illustrate the concepts of fault, error, failure, and the possibilities with in terms of their activation and propagation of faults and errors. **(7 points)**

   ```
   def add2(x):
       y = x * 2 # fault: should be x + 2
       if y == 6: # line A
           y -= 1 # line B
       return y
   ```

   When x is 2, fault is desensitized as 2*2 = 2+2.

   When x is 3, fault is sensitized (as 3*2 != 3+2) but error is masked (as the expected value 5 is returned) due to lines A and B.

   When x is 1, fault is sensitized (as 1*2 != 1+2) and error is propagated (as 2 is returned instead of 3).

3. What is the difference between specification, program, and test? **(3 points)**

   A **specification** describes what is to be accomplished, a **program** describes how something is accomplished, and a **test** checks parity between a specification and a program.

4. Define fault by commission and fault by omission. Provide an example for each kind of faults. **(4 points)**

   Fault by commission is a fault stemming from incorrect implementation, e.g., using x*2 instead of x+2. **Fault by omission** is a fault stemming from absence of correct implementation, e.g., using x%3 instead of (x*2)%3.

5. What are the differences between system testing and acceptance testing. **(4 points)**
   System testing is a development-team centric task/activity to test the whole system to uncover faults before releasing it to the user. **Acceptance testing** is a user centric task/activity to test the whole system to ensure it satisfies the requirements.
6. What are the necessary elements/abilities to realize any kind of testing? **(3 points)**
   - Knowledge of the expected outcome,
   - ability to observe the concerned outcome, and
   - ability to compare outcomes.
7. List four aspects of testing that can be automated to make testing better (and easy and simple). **(4 points)**
   - Test data generation
   - Test case generation
   - Test execution
   - Test result collection and compilation
8. Under what situations should we use assertions in programs (UUT)? Provide an example situation where assertions are appropriate in programs. **(3 points)**
   Use assertions in UUT to check inputs (data) are valid; when only valid inputs are expected (or the client will ensure the inputs are valid), e.g., when sq_root should be invoked only with positive integers, we can use assert in sq_root to check the input is a positive integer.
9. What is (typically) done in test fixture methods? **(3 points)**
   - Put the UUT in the state required for testing,
   - Perform actions common to a set of test cases, and
   - Perform clean up actions.
10. What is the typical structure of parameterized unit tests? **(5 points)**
    - Setup,
    - Check assumptions about parameters,
    - Execute,
    - Verify, and
    - Teardown

11. Write example-based unit tests in Python to test the function `sorted(list_of_ints, descending)` that sorts a list of integers. The function expects `list_of_ints` to be a list of integers and `descending` to be a boolean value that if `True` indicates the list should be sorted in descending order. The function will raise a ValueError if the inputs are not as expected (invalid). **(10 points)**

```
def test_first_param_not_list():
    nt.assert_raises(ValueError, sorted, “”, True)

def test_first_param_invalid_list():
    nt.assert_raises(ValueError, sorted, [‘a’, ‘b’], True)

def test_second_param_is_invalid():
    nt.assert_raises(ValueError, sorted, [1, 2], “”)

def test_ascending_sort():
    nt.assert_equals(sorted([4,2,3], False), [2,3,4])

def test_descending_sort():
    nt.assert_equals(sorted([4,2,3], True), [4,3,2])
```

12. Complete the following parameterized unit test (in Python) with verification steps to test if the `sorted` function in Q11 sorts a list of integers in ascending order. **(15 points)**

```
import collections
def test_sorted_ascending(list_of_ints):
  result = sorted(list_of_ints, False)
  # your verification steps go here

  # test for ordering
  for i in range(0, len(result) - 1):
    assert result[i] <= result[i+1]

  # test for elements and their frequency
  tmp1 = collections.Counter(result)
  tmp2 = collections.Counter(list_of_ints)
  assert tmp1 == tmp2
```

13. Generalize your above parameterized unit test to test if the `sorted` function in Q11 sorts a list of integers in the given order. **(5 points)**

```
import collections
def test_sorted_ascending(list_of_ints, descending):
  result = sorted(list_of_ints, descending)
  # your verification steps go here

  # test for ordering
  for i in range(0, len(result) - 1):
    if descending:
      assert result[i] >= result[i+1]
    else:
      assert result[i] <= result[i+1]

  # test for elements and their frequency
  tmp1 = collections.Counter(result)
  tmp2 = collections.Counter(list_of_ints)
  assert tmp1 == tmp2
```