Term Test 1

COSC 4313 3.0 Software Engineering Testing

Section M, Winter 2008

Family Name: _____

Given Name(s): _____

Student Number: |___ |___ |___ |___ |___ |___ |___ |___ |___ |

Question	Out of	Mark
Q1	20	
Q2	20	
Q3	30	
Q4	30	
Total	100	
Letter grade		

1. **[20 marks]** In this question, you will be deriving test cases for a piece of software that helps a car insurance company assign risk categories and surcharges to a particular driver. There are three inputs to the program: Gender, Marital Status, and Age group: A (less than 21), B (21 to 25), C (26+). The rules are:

   If the applicant is under 21, apply a surcharge. If the applicant is male and under 26 and married, or male and over 26, assign him to risk category B. If the applicant is a single male under 26 or a female under 21, assign the applicant to risk category C. All other applicants are assigned to risk category A.

   Your task is to describe the application of any software testing strategies that you believe are appropriate in order to test this program. The derived test cases must be identified clearly in your answer. Answers that list test cases without describing how they were derived will receive a poor mark.

The complex logic in this problem indicates that a decision table is the right approach. Here are all 12 combinations [10 marks]:

Rule #	1	2	3	4	5	6	7	8	9	10	11	12
Age	A	A	A	A	B	B	B	B	C	C	C	C
Gender	M	M	F	F	M	M	F	F	M	M	F	F
Married	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N
Surcharge	X	X	X	X								
Risk A							X	X			X	X
Risk B	X				X				X	X		
Risk C		X	X	X		X						

The following sets of rules have the same action set and two of the three inputs common: 7-8, 9-10, 11-12. In each case the table can be reduced by one rule. Rules 2-4 also have the same action set. However, only of two of the three can be combined, i.e. either 2 and 4, or 3 and 4. The following table shows the first possibility, though either is sufficient for full marks. Rule numbering is retained for demonstration purposes [10 marks].

Rule #	1	2	3	5	6	7	9	11
Age	A	A	A	B	B	B	C	C
Gender	M	-	F	M	M	F	M	F
Married	Y	N	Y	Y	N	-	-	-
Surcharge	X	X	X					
Risk A						X		X
Risk B	X			X			X	
Risk C		X	X		X			

2. **[20 marks]** Consider the following function that prints the first n primes.

```
1  private static void printPrimes (int n)
2  {
3    int curPrime; // Value currently considered for primeness
4    int numPrimes; // Number of primes found so far.
5    boolean isPrime; // Is curPrime prime?
6    int [] primes = new int [MAXPRIMES]; // The list of prime numbers.
7
8    // Initialize 2 into the list of primes.
9    primes [0] = 2;
10   numPrimes = 1;
11   curPrime = 2;
12   while (numPrimes < n)
13   {
14     curPrime++; // next number to consider ...
15     isPrime = true;
16     for (int i = 0; i <= numPrimes-1; i++)
17     { // for each previous prime.
18       if (isDivisible (primes[i], curPrime))
19       { // Found a divisor, curPrime is not prime.
20         isPrime = false;
21         break; // out of loop through primes.
22       }
23     }
24     if (isPrime)
25     { // save it!
26     primes[numPrimes] = curPrime;
27     numPrimes++;
28     }
29   } // End while
30
31   // Print all the primes out.
32   for (int i = 0; i <= numPrimes-1; i++)
33   {
34     System.out.println ("Prime: " + primes[i]);
35   }
36 }
```

Draw the Control Flow Graph for this function. Clearly indicate what are the segments (you can do this on the code if you prefer).

The 12 segments for printPrimes (each worth 1 mark) are shown below. The Control Flow Graph is shown next (8 marks). Each incorrect edge is minus two marks. Segment Z denotes the end of the execution of printPrimes.

Segment A: Lines 1-11

Segment B: Line 12

Segment C: Lines 14-15 plus the initialization of the for loop.

Segment D: `i <= numPrimes - 1`

Segment E: Lines 17-18

Segment F: Lines 20-21

Segment G: `i++`

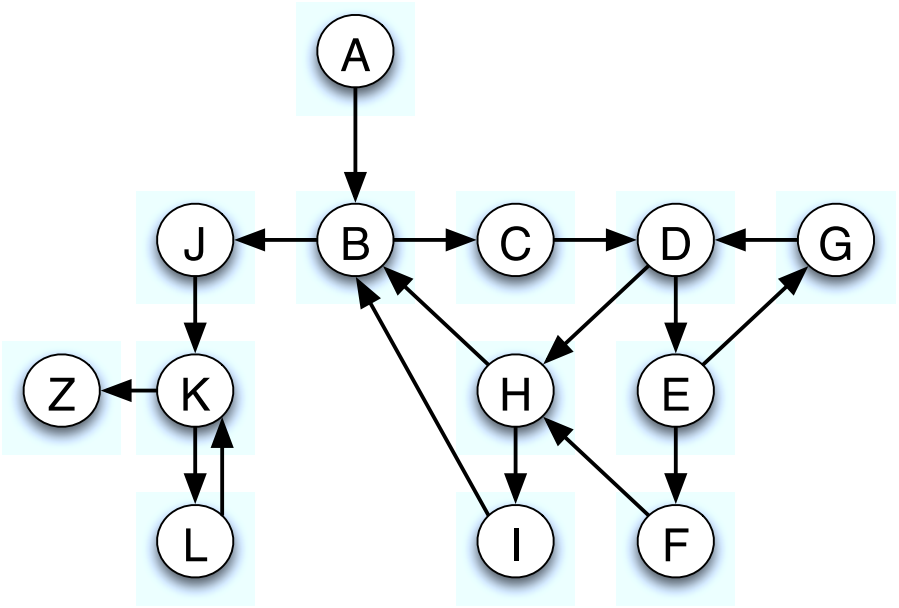
Segment H: Line 24

Segment I: Lines 26-27

Segment J: The initialization od the for loop in line 32

Segment K: `i <= numPrimes - 1`

Segment L: Line 34 plus `i++`



3. **[30 marks]** For the function of the previous question, indicate and explain briefly how you derived

   (a) A minimal test suite that achieves 100% segment coverage [10 marks].

   Segments A and B will be covered with any test case. Similarly, any test case where n is larger than 0 will cover segments J, K, and L. A value for n larger than 1 will ensure that we enter the loop, which will cover segments C,D,E,G,H,I (notice that we cannot exit the loop unless we visit segment I). So, we simply need to select a test case that will take us to segment F. This means that as we are searching for prime numbers, we encounter a number that is not prime. For this to happen, we need n to be 3. This will discover the prime numbers 2, 3, and 5, and will pass over 4.

   Therefore, our minimal test suite consists of one test case, n=3.

   (b) A minimal test suite that achieves 100% branch coverage [10 marks].

   The test suite that achieves segment coverage, also achieves branch coverage, so there is no need for additional test cases.

(c) The value of the All-Uses coverage criterion with respect **only** to the `numPrimes` variable for a test suite containing only one test case, n=100 [10 marks].

There are two definitions for `numPrimes` in lines 10 and 27. There are also 5 uses for it in lines 12, 16, 26, 27, and 32. As a result, we are looking for 10 d-c paths that our test suite should cover.

Definition in line 10: The value of 1 assigned there will be used throughout the first iteration of the while loop, so the uses in lines 12, 16, 26, and 27 will be covered. However, it is clear that the use in line 32 will never use the definition in line 10. As a result, 4 out of 5 paths are covered.

Definition in line 27: Similarly, all the uses inside the while loop will use the incremented value in the next iteration. Also, the last time the value is incremented, the loop will exit, and we will use this value on line 32. As a result, 5 out of the 5 paths are covered.

Therefore, the coverage with respect to the All-Uses criterion is 9 out of 10, or 90%.

4. **[30 marks]** Consider the following specification for a software system.

   MTEST is a program that grades multiple-choice examinations. Input and output are data files.

   The first line in the input file contains an integer between 1 and 999. This is the number of questions in the exam. The next few lines describe the correct answers for the exam. Each one of these lines starts with 9 zeros followed by answers for 10 questions (each answer is a capital letter). The last line of correct answers may have less than 19 characters depending on the number of questions.

   This is followed by the answers of every student that took the exam. The format is the same as for the correct answers, except the 9 zeros are replaced by the student number of the student that provided the answers.

   The output file contains a line per student. The line starts with the student number, followed by a space, followed by the number of questions the student answered correctly. The output file is sorted by ascending student number.

   The following is an example of corresponding input and output files:

```
15
000000000ABCDEFGHIJ
000000000KLMNO
234675643ABCDEFGHIJ
234675643KLMNO
2343252333ZZZZZZZZZZ
2343252333ZZZZZ
```

```
234325233 0
234675643 15
```

   Your task is to describe the application of any software testing strategies that you believe are appropriate in order to test this system. The derived test cases must be identified clearly in your answer. Answers that list test cases without describing how they were derived will receive a poor mark.

   The specification above is provided as is. You may have to make assumptions about system behaviour that is not mentioned in the specification. If this is the case, choose reasonable ones and state them explicitly.

The testing strategy that fits this system is a combination of Boundary Value Testing with Equivalence Class Testing.

An aspect of the system that needs to be tested is input validation. We can tackle this by identifying equivalence classes for invalid inputs, such as:

IV1: The first line does not contain an integer between 1 and 999.

IV2: Subsequent lines do not begin with 9 digits.

IV3: The number of lines starting with 0s is incorrect

IV4: Student entries contain incorrect number of answers.

These classes can be organized differently, and they can also be refined further.

Assuming a valid input, there are a number of input variables to consider:

(a) File size: Try empty file, a file containing just a 0, a very large file.

(b) Number of questions: There are two equivalence classes to consider, since there is a boundary at 10 questions when we switch from single line entries to multiple line entries per student. At a minimum one should try test cases with 1, 10, 11, and 999 test cases.

(c) Number of students: Try 0,1,MAXINT number of students.

(d) Number of correct answers: Try student entries with 0, 1, n-1, n correct answers, where n is again 1, 10, 11, 999 (two equivalence classes again).

Testing for sorted output should also be done. However, there is no need to develop separate test cases since all the above test cases will have expected output that is sorted.