CSCE 747 - Assignment 2:
Structural and Data Flow Testing

Due Date: Tuesday, February 23rd, 11:59 PM

There are 5 questions, worth a total of 100 points. You may discuss these problems in your teams and turn in a single submission for the team, in PDF format, on Moodle. Answers must be original and not copied from online sources.

Problem 1 (5 Points)

Answer exercise 9.3 from the textbook.

Problem 2 (40 Points)

Consider the C program provided at the end of this document. It is an implementation of a weighted-round-robin (WRR) resource allocation algorithm. The program takes the number of clients (n) and their individual weights, $\{w_i \mid 1 \leq i \leq n\}$ as input and determines a weighted-round-robin schedule for allocating some shared resource to the clients.

The output is a sequence of (repeated) client numbers (from the range 1….n) to be allocated in order, one unit of the shared resource.

1. Show the control-flow graphs for the three procedures in the program.
2. Show the call graph for the program. Include the library functions in your call graph.
3. Provide a test suite that achieves statement coverage, but not branch coverage.
4. Provide a test suite that achieves branch coverage, but not condition coverage. (Treat compound conditions as a single branch point for this purpose.)
5. Provide a test suite that achieves condition and branch coverage. What additional test cases, if any, do you need for MC/DC coverage?
6. Do the above test suites (taken together) achieve loop boundary coverage for all the loops in the program? If yes, identify the test cases that achieve this for each of the loops. If not, provide additional test cases to achieve loop boundary coverage (zero, one, and many times through the loop).

If any of these coverage metrics have infeasible test obligations, justify why that is the case. Feel free to try executing the code (no guarantees here), with any additions that you may need, to check whether your tests achieved the desired level of structural coverage.

Problem 3 (10 Points)

Exercise 6.4 from the textbook.

Problem 4 (25 Points)

In a directed graph with a designated exit node, we say that a node m post-dominates another node n, if m appears on every path from n to the exit node.

Let us write *m pdom n* to mean that m post-dominates n, and *pdom(n)* to mean the set of all post-dominators of n, i.e., *{m | m pdom n}*.

Answer the following, providing justification for each:
1. Does *b pdom b* hold true for all b?
2. Can both *a pdom b* and *b pdom a* hold true for two different nodes a and b?
3. If both *c pdom b* and *b pdom a* hold true, what can you say about the relationship between c and a?
4. If both *c pdom a* and *b pdom a* hold true, what can you say about the relationship between c and b?
5. How would you use the answer to your previous question to characterize the immediate post-dominator of a node other than the exit node?
(Hint: immediate post-dominator is, in some sense, the “nearest post-dominator” of a node)
6. Suppose the set of post-dominators *pdom(n)* is known for every successor n of a node m, can we then derive *pdom(m)*?
7. Cast the computation of *pdom(m)* in terms of flow equations. What are the gen and kill sets for m? How would you classify the flow analysis: forward or backward, any-path or all-path?
8. Provide an iterative work-list algorithm (see Chapter 6 for examples) for computing the post-dominance relation based on the flow equations.

Problem 5 (20 Points)

The following code is a Java function that does insertion sort.

```
public static void InsertionSort(int[] num)
{
	int j;   // the number of items sorted so far
	int key; // the item to be inserted
	int i;

	for (j = 1; j < num.length; j++)    // Start with 1 (not 0)
	{
		key = num[j];
		// Smaller values are moving up
```

```
            for(i = j - 1; (i >= 0) && (num[ i ] < key); i--)
            {
                  num[i+1] = num[i];
            }
            num[i+1] = key;     // Put the key in its proper location
      }
}
```

1. Identify the def-use pairs for all the variables. Explain how you handle the array variable. *Hint: arrays are pointers in Java.*
2. Provide a test suite that achieves all def-use pairs coverage. If there are def-use pairs that cannot be covered, explain why.

```
#include <stdio.h>
#include <stdlib.h>
#define SWAP(a,b) { int _tmp = a; a = b; b = _tmp; }

int cur_count = 0, alloc_count = 0, total_count = 0, num_elems = 0;
int *counts, *psums, *elems;

void initialize(int n, FILE *fptr) {
      int i = 0;
      num_elems = n;
      counts = (int *) calloc(n, sizeof(int));
      psums = (int *) calloc(n, sizeof(int));
      elems = (int *) calloc(n, sizeof(int));

      for (i = 0; i < n; ++i) {
            elems[i] = i + 1;
            fscanf(fptr, "%d", &counts[i]);
            psums[i] = total_count += counts[i];
      }
}

int next_elem() {
      int top = num_elems - 1;
      int ret = elems[top];

      if (alloc_count >= total_count)
            return 0;

      ++alloc_count;
      ++cur_count;
      --psums[top];
      --counts[top];

      while (top > 0 && counts[top] <= (cur_count * psums[top - 1])) {
            psums[top - 1] += (counts[top] - counts[top - 1]);
            SWAP(counts[top], counts[top - 1]);
            SWAP(elems[top], elems[top - 1]);
            --top;
      }

      if (top != num_elems - 1)
            cur_count = 0;

      return ret;
}
```

```
int main() {
	int next;
	printf("number of elements: ");
	scanf("%d", &num_elems);

	printf("\nweights:\n");
	initialize(num_elems, stdin);

	printf("\n");
	while ((next = next_elem()) > 0)
		printf("%d ", next);
	printf("\n");
}
```