University of Toronto
CSC148 – Introduction to Computer Science, Summer 2003

Mid Term Test – Section 5101

Duration: 50 minutes
Aids allowed: 1 8.5” x 11” piece of paper with information written on one or both sides.

Make sure that your examination has 6 pages (including this one). Write your answers in the spaces provided. Write legibly. You may use page 6 for rough work (tear it off, if you like). If you require more space to answer a question, write on the back of the previous page, and indicate in the answer space where your answer is.

Personal Information:

Surname:	
Given name(s):	
Student #:	
Circle section of registration:	**L0101 or L5101**

Please note that if you write the midterm in pencil, you will ***not*** be allowed to submit a remark request.

Do not write anything on this page below this line:

1.	/ 12
2.	/ 12
3.	/ 6
Bonus.	/ 0
Total:	/ 30

Question 1: Quick Concept Questions

[3 sub-questions, 4 marks each]

a. Briefly explain the difference between a `null` String (eg: `String s = null;`) and an empty String (eg: `String s = "";`). Include a description of the differences seen in the Object Space of computer memory.

A null String stores the value "null" in the memory space for the reference, and contains no reference to anything in Object Space. An empty string is a reference to an instance of the String class in Object Space that is storing the value "".

3 marks for stating empty is an instance in the object space, null is not
1 mark for stating that s is a reference to that empty string object or contains "null"

b. Below is the skeleton of a try/catch block:

```
try {
   // block 1: code that throws an exception
} catch (ExceptionClass1 e) {
   // block 2: code that deals with this type of exception
} catch (ExceptionClass2 e) {
   // block 3: code that deals with this type of exception
}
```

An exception thrown in block 1 might match either ExcpetionClass1 or ExceptionClass2. Under what circumstance would it match both? Which block of code (2 or 3) would run if this were to happen?

If ExceptionClass1 extends ExceptionClass2 (or vice versa), and the thrown exception is the sub-class, then it matches both. Block 2 would be the one to run.

3 marks for saying one of EC1 or EC2 extends the other (or exists in same path in inheritance hierarchy tree)
1 mark for saying that more specific (sub-class) must be thrown

c. If a tree has a branching factor of 5, what is the *minimum* number of child nodes a node in that tree may have?

0 (any tree with at least one node must have a leaf, which has no children)

4 marks for simply writing "0"

Question 2: Linked List

[12 marks]

Here is a node class:

```
class IntNode {
   int data;
   IntNode next;

   IntNode(int data, IntNode next) {
      this.data = data;
      this.next = next;
   }
}
```

Complete the method body for "replaceMin" below. ***Be sure to include all internal comments.***

```
/* Replace the minimum value in the list rooted at head with the
 * value n.
 *
 * Requires:
 *    There is at least 1 element in the list rooted at head.
 *    There are no duplicate values in the list rooted at head */

public static void replaceMin(IntNode head, int n) {


      // Algorithm: traverse the list, maintaining a reference to
      // the node with the minimum value.  Replace that value with n.

      IntNode min = head;
      IntNode next = head.next;

      // find the min by comparing it with every element in the list
      while (next != null) {
         if (min.data > next.data) {
            min = next;
         }
         next = next.next;
      }

      // replace its data with n
      min.data = n;

   1 mark for algorithm comment at top of method
   1 mark for comments throughout method explaining correct
      algorithm at a high level
   4 marks for correctly traversing the list, finding the min,
      and replacing it with n.



} // did you remember to include comments?
```

Complete the method body for "insertAt" below (using the IntNode class from the previous page). ***Be sure to include all internal comments.***

```
/* Inserts "n" into the list rooted at "head" before the ith
 * node in the list (counting from 0).  Returns the head of the
 * list containing n.
 *
 * Requires: there at least i+1 nodes in the list rooted at head
 */

public static IntNode insertAt(IntNode head, int i, int n) {


      // algorithm: traverse the first i elements of the list, keeping
      // a reference to the i-1th element.  Insert a new node into the
      // list at that spot.  If i==0, insert before the head.

      // insert the new value before the head, it becomes the new head.
      if (i==0) {
         head = new IntNode(n,head);
      } else {
         // find the i-1th element, insert the new node after it

         IntNode next = head;

         // find the i and i-1th element
         for(int j=0; j < i; j++) {
            next = next.next;
         }

         next.next = new IntNode(n,next.next);

      }

      // return the (possibly new) head of the list
      return head;

   1 mark for algorithm comment at top of method
   1 mark for comments throughout method explaining correct
      algorithm at a high level
   2 marks for identifying and dealing correctly with i=0 case
   2 marks for identifying and dealing correctly with more
      general case (i > 0)





} // did you remember to return something, and include comments?
```

Question 3: Recursion

[6 marks]

Here is a recursive method:

```
/* Finds and returns the value of n factorial (n!).
 * Requires: n>= 0
 */

int factorial(int n) {

   if (n==0) {
      return 1;
   } else {
      // A: return factorial(n);
      // B: return n * factorial(n);
      // C: return n * factorial(n--);
      // D: return n * factorial(--n);
   }
}
```

In order for the factorial method to work as specified, which line of code should be uncommented? Circle the correct answer:

The answer is D 4 marks for circling the correct answer

A ***B*** ***C*** ***D***

Bonus question

[+3 marks]

Would the line:

```
return (n--) * factorial(n);
```

have worked in place of the commented lines above? Circle one of:

YES ***NO***

Explain your answer:

Because the value of "n--" is "n", but the effect is to decrease n by one, the above line is effectively equivalent to: return (n) * factorial(n-1);

The recursive call to factorial will return the multiple of [1,…,.n-1]. We need only multiply this by "n" to it to yield the multiple of [1,…,n] (n!)

0 marks for circling “yes”

3 marks for correctling explaining that the above line is equivalent to “return n * factorial(n-1)”

This page intentionally left blank. Tear it out and use it for scratch paper.