SYSC 2100, Fall 2006 Midterm
Sample Solutions

Question 1. Recursion (10 marks)

1. Write a recursive Java method *writeLine* that writes a character repeatedly to form a line of *n* characters. For example, *writeLine('*', 5)* produces the line

   *****

   Answer (5 marks):

```
void writeLine(char ch, int n)
// ---------------------------------------------------------
// Outputs a line of n characters where ch is the character.
// Precondition:   n >= 0.
// Postcondition:  A line of n characters ch is output
//                 followed by a newline.
// ---------------------------------------------------------
{  // base case
   if(n <= 0)
      System.out.println();

   // write rest of line
   else
   {  System.out.print(ch);

      writeLine(ch, n - 1);
   } // end if
} // end writeLine
```

2. Write a recursive method *writeBlock* that uses writeLine to write *m* lines of *n* characters each. For example, *writeBlock('*', 5, 3)* produces the output

   *****
   *****
   *****

   Answer (5 marks):

```
void writeBlock(char ch, int m, int n)
// --------------------------------------------------------
// Outputs a block of m rows by n columns of character ch.
// Preconditions:  m >= 0 and n >= 0.
// Postconditions:  A block of m rows by n columns of
//                  character ch is printed.
// --------------------------------------------------------
{  if(m > 0)
   {
      writeLine(ch, n);              // write first line
      writeBlock(ch, m - 1, n);      // write rest of block
   }

   // else m <= 0, do nothing (base case).
} // end writeBlock
```

Question 2. ADT List (10 marks)

1. In the discussion in class, we described methods *displayList* and *replace* for the ADT List. As described, these methods exist outside of the ADT; that is, they are not operations of the ADT. Instead, their implementations are written in terms of the ADT's operations.

   a. What is an advantage and a disadvantage of the way *displayList* and *replace* are implemented?

   Answer (2 marks):
   A clear advantage to defining such operations externally to the ADT is the control that the client has in customizing the functionality. For example, the manner in which the list items are displayed can be adjusted or formatted or an additional test for replacement of a list item may be performed at the client's discretion.

   The disadvantage of this implementation is that the client may fail to test such important procedures as indexing past the end of the list or performing the insertion into the list after the deletion fails.

   b. What is an advantage and a disadvantage of adding the operations *displayList* and *replace* to the ADT?

   Answer (2 marks):
   Defining these operations within the ADT obviously alleviates the disadvantage cited in part a. Proper bounds checking and other testing is performed within the ADT, removing such concerns from the client.

   However, the client cannot control how these operations perform and if greater refinement of control is required he must write additional functionality making use of the suite of operations provided by the ADT.

2. Write a method to merge two linked lists of integers that are sorted into ascending order. The result should be a third linked list that is the sorted combination of the original lists. Do not destroy the original lists.

Answer (6 marks):

```
void mergeSortedLists(Node head1, Node head2, Node mergeHead)
// ------------------------------------------------------------------
// Merges two lists sorted in ascending order into a third list.
// Preconditions:  The lists referenced by head1 and head2 are sorted
//     in ascending order.  mergeHead is an initialized Node.
// Postconditions:  mergeHead references a sorted list containing the
//     contents of the lists referenced by head1 and head2 in ascending
//     order.  The original lists referenced by head1 and head2 are
//     unchanged.  Anything previously referenced by mergeHead is lost.
// ------------------------------------------------------------------
{  boolean firstNode = true;
   Node cur1 = head1, cur2 = head2;
   Node      mergeCur = new Node();

   // if neither list is empty merge them
   while(cur1 != null && cur2 != null)
   {  if(firstNode)
      {  mergeCur = mergeHead;
         firstNode = false;
      } // end if
      else
      {  mergeCur.setNext(new Node());
         mergeCur = mergeCur.getNext();
      } // end else

      mergeCur.setNext(null);    // terminate the list

      if(cur1.getItem() < cur2.getItem())   // copy item from list 1
      {  mergeCur.setItem(cur1.getItem());
         cur1 = cur1.getNext();
      } // end if
      else                              // copy from list 2
      {  mergeCur.setItem(cur2.getItem());
         cur2 = cur2.getNext();
      } // end else

   } // end while

   if(cur1 != null)  // determine which list is not completely traversed
      cur2 = cur1;


   while(cur2 != null)
   {  if(firstNode)  // in case one of the lists was empty
      {  mergeHead = new Node();
         mergeCur = mergeHead;
         firstNode = false;
      } // end if
      else
      {  mergeCur.setNext(new Node());
         mergeCur = mergeCur.getNext();
      } // end else

      mergeCur.setItem(cur2.getItem());
      mergeCur.setNext(null);
      cur2 = cur2.getNext();
   } //end while

} // end mergeSortedLists
```

Question 3. ADT Stack (10 marks)

1. Write a pseudo-code method *isInL(s)* that uses a stack to determine whether a string *s* is in the language L, where L = {w: w is of the form A^nB^n for some n >= 0}

   Answer (4 marks):

```
isInL(s)
{  // Determines if the string s is in the language L.
   // Returns true if s consists of a number of A's followed by the
   // same number of B's.

   s.createStack()
   i = 0
   size = length of s


   // while there are A's in the string, push onto stack
   while(i < size and s.charAt(i) == 'A')
   {  s.push(s.charAt(I))
      i++
   }

   // should only be B's in rest of string
   while(i < size and !s.isEmpty and s.charAt(i) == 'B')
   {  s.pop()
      i++
   }

   // if A's and B's match, stack should be empty
   return (s.isEmpty() and i == size)
}
```

2. Suppose that you have a stack *aStack* and an empty auxiliary stack *auxStack*. Show how you can do each of the following tasks using only the operations of the ADT stack (i.e., write the pseudo-code for the following tasks):
   a. Count the number of items in aStack, leaving aStack unchanged.

   Answer (3 marks):

```
int countItems(StackReferenceBased s)
// Counts the number of items in the stack s.
{  Object stackItem;
   StackReferenceBased t = new StackReferenceBased ();
   t.createStack();
   int counter = 0;

   // put items into reverse order and count the items
   while(!s.isEmpty())
   {  stackItem  = s.pop();
      t.push(stackItem);
      counter++;
   } // end while

   // restore the original stack
   while(!t.isEmpty())
   {  stackItem  = t.pop();
      s.push(stackItem);
   } // end while

   return counter;
} // end countItems
```

   b. Delete every occurrence of a specified item from aStack, leaving the order of the remaining items unchanged.

   Answer (3 marks):

```
void removeItem(StackReferenceBased s, Object item)
// Removes all occurences of Item from the stack s.
{  Object stackItem;
   StackReferenceBased t = new StackReferenceBased ();
   t.createStack();

   // put items into reverse order removing all occurrences of Item
   while(!s.isEmpty())
   {  stackItem  = s.pop();
      if (stackItem!= item)
         t.push(stackItem);
   } // end while

   // restore the original order of the remaining stack items
   while(!t.isEmpty())
   {  stackItem  = t.pop();
      s.push(stackItem);
   } // end while
} // end removeItem
```

Question 4. Algorithm Efficiency and Sorting (10 marks)

1. Suppose that your implementation of a particular algorithm appears in Java as

```
for (int pass = 1; pass <= n; ++pass) {
      for (int index = 0; index < n; ++index) {
            for (int count = 1; count < 10; ++count {
                  …..
            } // end for
      } // end for
} // end for
```

The previous code shows only the repetition in the algorithm, not the computations that occur within the loops. These computations, however, are independent of n. What is the order of the algorithm? Justify your answer.

Answer (4 marks):
The algorithm is $O(n^2)$. Suppose the computation in the innermost for loop takes time proportional to some method *f(x)*, where *x* is independent of *n*. Then, the innermost loop must take, in the worst case, 10 * *f(x)* steps to compute. Since *x* is independent of *n* by hypothesis, the value 10 * *f(x)* reduces to some constant *c*. The overall computation then requires *c* passes for each of the *n* passes of the middlemost for loop and the middle loop requires *n* passes in turn for each of the *n* passes of the outermost loop. Thus the overall time complexity is *c*n*n* or $O(n^2)$.

2. Apply the selection sort, bubble sort, and insertion sort to
   a. An inverted array: 8 6 4 2
   b. An ordered array: 2 4 6 8

Show the resulting array after each change.

The inverted array 8 6 4 2
Answer (3 marks):
i selection sort

action	array
initial array	8 6 4 2
first swap	2 6 4 8
second swap	2 4 6 8
no swap	2 4 6 8

ii. bubble sort

action	array
initial array	8 6 4 2
pass 1	6 8 4 2
	6 4 8 2
	6 4 2 8
pass 2	4 6 2 8
	4 2 6 8
	4 2 6 8
pass 3	2 4 6 8
	2 4 6 8
	2 4 6 8

iii. insertion sort

action	array
initial array	8 6 4 2
copy 6 and shift 8	8 8 4 2
insert 6	6 8 4 2
copy 4 and shift 6, 8	6 6 8 2
insert 4	4 6 8 2
copy 2 and shift 4, 6, 8	4 4 6 8
insert 2	2 4 6 8

The sorted array 2 4 6 8

Answer (3 marks):

i selection sort

action	array
initial array	2 4 6 8
no swap	2 4 6 8
no swap	2 4 6 8
no swap	2 4 6 8

ii. bubble sort

action	array
initial array	2 4 6 8
pass 1	2 4 6 8
	2 4 6 8
	2 4 6 8

iii. insertion sort

action	array
initial array	2 4 6 8
copy 2 on itself	2 4 6 8
copy 4 on itself	2 4 6 8
copy 6 on itself	2 4 6 8