CSCE 146 – Midterm Exam 1 SAMPLE– 24 September 2008

You have **75 minutes. Look over the test first. Divide points into time to estimate how much time you ought to spend on any given question.**

Justify any answer you give.

You may wish to make simplifying assumptions. **STATE THOSE ASSUMPTIONS CLEARLY. If the assumptions are entirely reasonable (that is, they turn an otherwise very difficult problem into a problem that is reasonable for an exam) then this is perfectly acceptable. If your assumptions turn an otherwise reasonable problem into a relatively trivial problem, then I will not grant full marks for the answer.**

NAME:

1. (xx)	5. (xx)	9. (xx)
2. (xx)	6. (xx)	10. (xx)
3. (xx)	7. (xx)	
4. (xx)	8. (xx)	Total

Please note that I have given justifications for many of the statements here. You are expected to provide justifications in your answers. Simply making a statement, without justification, will not guarantee you full marks even if it happens to be the case that the statement is true.

1. (20 points total)

   (a) (5 points) Define what is meant by a **queue**.

   (b) (5 points) Why might one use a queue as a data structure?

   (c) Assume you are writing a class for a queue.

      i. (4 points) What are the instance variables that are required?
      ii. (6 points) What are the basic methods that are required?

Answer:

A queue is a data structure consisting of a list of "nodes" or data items in which additions to the list are made only at one end (the tail) and removals are made only at the other end (the head) so as to implement a FIFO process.

Because a queue implements a FIFO process, it is a natural structure for retail checkout lines, processing of service requests, etc. In a computer operating system, the set of processes capable of execution is usually organized in a queue and dispatch for execution is made with a circular round-robin scheduling system.

Assuming the individual nodes/data items are treated as independent objects, the only required instance variables would be a pointer to the head, a pointer to the tail, and perhaps a count of the number of entries in the queue.

Again assuming independent nodes, it is necessary to have a method to add an entry at the tail, a method to remove the entry from the head, and perhaps a "getNext" if it was necessary to traverse the list.

2. (15 points total)

   (a) (5 points) What makes a method a **recursive** method?

   (b) (10 points) Write a method to compute the factorial function $F(n)$ using recursion.

   Answer:

   If we choose not to worry about throwing exceptions, then:

```
public int factorial(int n)
{
  int returnValue;
  if(0 == n)
  {
    returnValue = 1;
  }
  else if(1 == n)
  {
    returnValue = 1;
  }
  else
  {
    returnValue = n * factorial(n-1);
  }

  return returnValue;
}
```

3. (5 points) Finish the definition:

   A function $f(x)$ is $O(g(x))$ if

   Answer:

   there exist constants c_0 and C_1 such that for all $x > c_0$, we have $|f(x)| \leq C_1 g(x)$.

4. (10 points) Prove FROM FIRST PRINCIPLES (that is, directly from the definition) that $x^3 - 2x^2 + 35$ is $O(x^3)$

Answer:

$|x^3 - 2x^2 + 35|$
$\leq |x^3| + |2x^2| + |35|$ ← because abs value distributes that way
$\leq |x^3| + |2x^3| + |35x^3|$ ← provided we have $x > 1$
$= 38|x^3|$ ← simple algebra

In the definition, we choose $c_0 = 1$ and $C_1 = 38$ and by definition we are done.

Note that this is truly a coarse analysis, and in spite of that this will work for all polynomials. This is essentially the argument used in the notes.

5. (10 points) What is the big Oh complexity of the following loop in terms of the number of elements in the array `abc[*]`?

```
int n = abc.length;
asztal = 4;
for(int i = 0; i < n; ++i)
{
  for(int j = 0; i < n/2; ++i)
  {
    asztal = asztal + i + j;
  }
}
```

Answer:

The `asztal = asztal + i + j` statement takes constant time K. The inner loop executes that statement $n/2$ times, so the cost of the inner loop is $K \times n/2$ "steps".

The outer loop executes the inner loop n times, so the cost of the outer loop is $n \times K \times n/2$ "steps".

The cost of the double loop is $(K/2)n^2$ steps, and by one of the basic theorems we can ignore the constant $K/2$ out front and assert that this is $O(n^2)$.

6. (10 points) Write the code fragment to traverse a linked list from a current node `myNode` through to the end of the list. You should assume the usual accessor and mutator methods.

    Answer:

    Starting from the current node `myNode`, the following loop will walk the list.

    ```
    while(null != myNode.getNext()) // we assume a getNext() method
    {
      myNode = myNode.getNext();
    }
    ```

7. (15 points) The following is a template for a class for a doubly linked list. Write the code for the `unlink` method.

    ```
    public class DLL implements IDLL
    {
      protected int size;
      protected DLLNode head, tail;

      public DLL()
      {
        this.head = null;
        this.tail = null;
        this.setSize(0);
      }

      public int getSize()
      {
      } // public int getSize()

      public void setSize(int value)
      {
      } // public void setSize(int value)

      public void add(DLLNode dllNode)
      {
      } // public void add(DLLNode dllNode)
    ```

```
/************************************************************************
 * Method to unlink a node.
 * @param node the 'DLLNode' to unlink.
**/
  public void unlink (DLLNode node)
  {
// CODE GOES HERE
  } // public void unlink(DLLNode node)

} // public class DLL implements IDLL
```

Answer:

```
  public void unlink (DLLNode node)
  {
    if(head != node)
    {
      node.getPrev().setNext(node.getNext());
    }
    else
    {
      head = node.getNext();
    }

    if(tail != node)
    {
      node.getNext().setPrev(node.getPrev());
    }
    else
    {
      tail = node.getPrev();
    }

    node.setNext(null);
    node.setPrev(null);

    this.decSize();

  } // public void unlink(DLLNode node)
```

8. (15 points) What does this code do?

```java
public class X
{
  int a,b,n;
  String[] z;

  public X(int nn)
  {
    n = nn;
    a = 0;
    b = 0;
    z = new String[nn];
  }

  public int getA()
  {
    return a;
  }

  public int getB()
  {
    return b;
  }

  public String yyy()
  {
    String s = z[a];
    a = (a+1) % n;
    return s;
  }

  public void xxx(String www)
  {
    b = (b+1) % n;
    z[b] = www;
  }
}
```

Answer:

The constructor creates an array of `String` data. The method `xxx` bumps pointer/subscript `b` by one modulo n and then stores an item of `String` data in the location pointed to.

The method `yyy` accesses the data pointed to by pointer/subscript `a` and then bumps that pointer modulo n.

Taken together, this implements a circular queue of `String` data, with the head being pointed to by `a` and the tail being pointed to by `b`.

Except for the error checking and bulletproofing, which is not done in this code.