CS 12 Fall 2003 – **Solutions for** mid-term exam #2

1. (10 points) **Compilers and interpreters**
   Provide short answers (a few sentences at most) to the following questions.

   (a) What is the difference between a compiler and an interpreter?

   An interpreter translates source code to machine code on demand, as the program runs. A compiler perform the translation once, before the program runs.

   (b) Is C++ compiled, interpreted, or both? Explain.

   C++ is compiled. The source code and header files are translated into a single executable file that contains the machine code. To execute a C++ program, the machine code from that executable file is read and processed directly by the CPU.

   (c) Is Java compiled, interpreted, or both? Explain.

   Java uses both. The source code is always compiled into bytecode. To execute the bytecode, the Java Virtual Machine (JVM) must translate that bytecode into machine code. It can do so either by interpreting the bytecode, or by compiling needed portions of the bytecode using a just-in-time (JIT) compiler.

2. (15 points) **Sorting**
Write pseudocode that would perform bubble-, selection-, or insertion-sort (*select one*) on an array of integers.

```
void bubbleSort (int[] array) {

  boolean swap;
  do {
    swap = false;
    for (int i = 0; i < array.length - 1; i++) {
      if (array[i] > array[i+1]) {
        int temp = array[i];
        array[i] = array[i+1];
        array[i+1] = temp;
        swap = true;
      }
    }
  } while (swap);

}

void selectionSort (int[] array) {

  for (int i = 0; i < array.length - 1; i++) {
    int minIndex = i;
    for (int j = i + 1; j < array.length; j++) {
      if (array[j] < array[minIndex]) {
        minIndex = j;
      }
    }
    int temp = array[i];
    array[i] = array[minIndex];
    array[minIndex] = temp;
  }

}

void insertionSort (int[] array) {

  for (int i = 1; i < array.length; i++) {
    int insertionValue = array[i];
    int j = i - 1;
    while ((j >= 0) && (insertionValue < array[j])) {
      array[j+1] = array[j];
      j--;
    }
    array[j+1] = insertionValue;
  }

}
```

3. (25 points) **Linked lists and arrays**
Consider a class that stores a set of Objects. This class provides a `find()` method that finds and returns a pointer to a given object. The class also provides a `getRecencyOrder()` method that indicates how *recently* each object was accessed through the `find()` method.
We want the class to keep a data structure that orders the objects based on how recently each has been accessed. To do so, we use the *move-to-front (MTF)* algorithm: When `find()` is called, the object is found in the data structure and then moved to the beginning of the order.
Answer the following questions about implementing the MTF algorithm with an array or linked list:

   (a) How long does a single MTF operation take using each choice? Express your answers with Big-O notation.

   Using an array, moving an object to the front takes O(n) time, since other objects must be copied to fill in the space left by removing the object from its original location. Using a linked list, this operation requires O(1) time, since removing a link takes a constant number of operations, and inserting at the front does also. If we include the time required to find the requested object, however, both arrays and linked lists require O(n) time to perform the search.

   (b) Consider that the class also keeps a *lookup table*—a structure that allows it to find a particular object within the ordering in O(1) time. Now how long does an MTF operation take using each choice of data structure? Again, use Big-O notation.

   The MTF operation itself is unaffected. However, if we include the time to search for the object, arrays still require O(n) as described above, while linked lists now require a total of O(1) time.

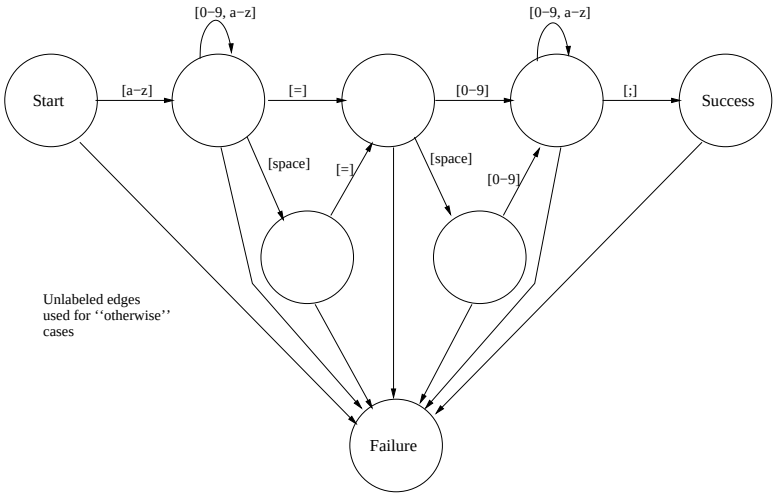
4. (25 points) **Parsing with state machines**
Consider a Java-like statement in which a variable is assigned a positive integer value—specifically, a line of the following form:

```
myvar15 = 3242;
```

For this statement, the following rules apply:

- Variable names are case insensitive.
- The variable name must start with a letter.
- Subsequent characters in a variable name may be letters or digits.
- The space following the variable name is optional.
- The equals sign appears between the variable name and the integer value.
- The space following the equals sign is optional.
- The integer may be any number of digits in length, and may only be positive.
- A semicolon must immediately follow the last digit of the integer.

Draw a state machine that will parse this statement. Clearly label the initial, failure, and success states. Also clearly label all transitions.



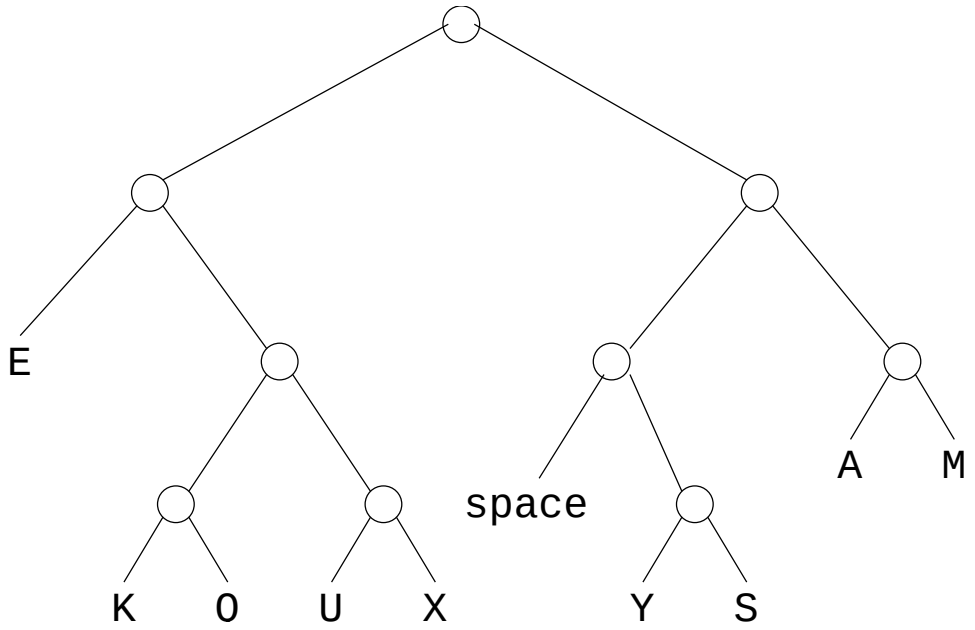
5. (25 points) **Compression**
Consider the following message:

```
EXAMS MAKE ME QUEASY
```

Each symbol occurs the following number of times in the message:

```
A       3
E       4
K       1
M       3
Q       1
S       2
U       1
X       1
Y       1
<space> 3
```

Encode this message using a Huffman code based on the symbol frequencies above. Be sure to provide a key or a tree that shows the bit pattern corresponding to each encoded symbol. *In the tree below, following the left edge corresponds to a `0`, and following a right edge corresponds to a `1`.*



Therefore, the encoded message is:

```
E  00                  M  111
X  0111                E  00
A  110                    100
M  111                 Q  0101
S  1011                U  0110
   100                 E  00
M  111                 A  110
A  110                 S  1011
K  0100                Y  1010
E  00
   100
```