CSE 326: Data Structures
Assignment #6
November 18, 1998
due: Wednesday, December 2

1. Read Section 5.4 through page 148. Keep your eyes peeled for any hint of a priority queue.
2. Implement the PriorityQueue class using heaps. You may assume that the heap never has more than 500 entries. You should assume that the Key field has type `double`, and the Info field is a pointer to a binary tree. DeleteMin should return a record containing both the minimum Key and its associated Info field.
3. Implement the following procedures that do Huffman encoding and decoding:
   (a) BuildHuffman (described in the paragraph starting on page 146 and ending on page 147): this reads a file of character-frequency pairs, and returns a Huffman decoding tree and encoding table built from them. Note that the leaves of your tree must each store a `char`. Unlike Figure 5.8, frequencies do not need to be stored in the tree. (Why?)
   Unlike the book's description, you should assume that the frequencies have type `double`: character frequencies are often given as percentages. Figure 1 gives the character-frequency pairs for you to use.
   These are the only 28 characters you need handle, except that your program should allow upper case and lower case letters interchangeably. Your program must read these character-frequency pairs from a file, because in practice these frequencies might be changed later, or the set of characters might be changed. Your program may assume that there will never be more than 500 characters. We will provide a file containing the pairs above.
   In addition to the decoding tree, BuildHuffman should return an encoding table, which is indexed by characters and whose entries give the encodings of those characters. This table can be constructed by a single traversal of the decoding tree.
   (b) TreeEncode (described in the first two sentences of the first full paragraph on page 146): this takes as inputs an encoding table and a stream w of characters, and outputs the stream of bits encoding w.
   (c) TreeDecode (described in Algorithm 5.2 on page 147): this takes as inputs an encoding tree and a stream b of bits, and outputs the character stream decoding b.

As described above, you will use character streams (for the decoded string) and bit streams (for the encoded string) as inputs and outputs. Character streams are easy: you can just use the standard I/O streams, reading characters from the keyboard and displaying characters to the screen.

Bit streams are slightly more complicated. Since the whole point of encoding is to compress characters, it would make no sense if your TreeEncode function outputs the chars '0' and '1', since this would use 8 bits for each bit in the encoding. We will provide a package that will handle the bit streams for you.

Instructions for your driver, bit streams, and turn-in will follow by electronic broadcast.

‘ ’	.200
‘.’	.010
‘a’	.073
‘b’	.009
‘c’	.030
‘d’	.044
‘e’	.130
‘f’	.028
‘g’	.016
‘h’	.035
‘i’	.074
‘j’	.002
‘k’	.003
‘l’	.035
‘m’	.025
‘n’	.078
‘o’	.074
‘p’	.027
‘q’	.003
‘r’	.077
‘s’	.063
‘t’	.093
‘u’	.027
‘v’	.013
‘w’	.016
‘x’	.005
‘y’	.019
‘z’	.001

Figure 1: Common English Character Frequencies