Cosc 1p03 Assignment 2

(Due date Feb 12th 16:00 est, Late date Feb 15th 16:00 est)

Background:

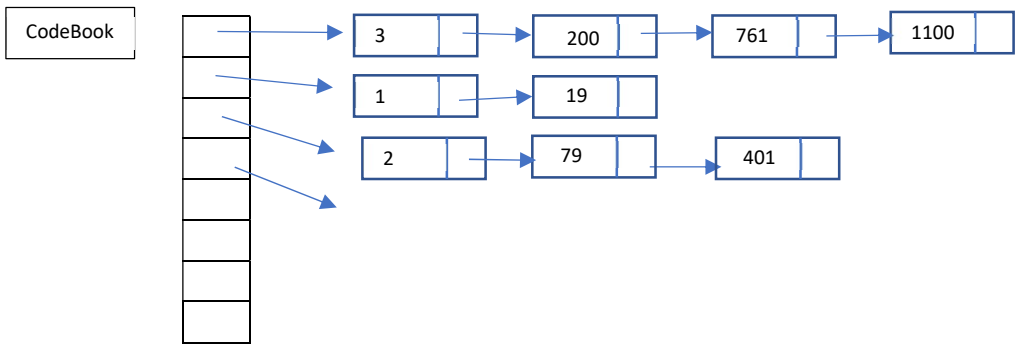
A Book Cypher [https://en.wikipedia.org/wiki/Book_cipher] is an encoding technique (encryption). A simple implementation in the days of books and pens was to take a passage from a book and sequentially number the words or letters in a chapter of a book. A message encoding would then select a corresponding number which represented a character or word from the cypher book. What was transmitted, was a series of numbers which could only be decrypted if the receiver had the same cypher book.

The Assignment

Let us implement a variation of this. For the purpose of this assignment we will randomly generate the code book and simulate an encoding and decoding as proof of concept. The standard character set known as ASCII [http://www.asciitable.com/] has 128 characters is the set, numbered from 0 to 127. For example, a blank character ' ' is character 32, a carriage return is character 13. Thus, to encode any message it would make sense to have this minimal set represented. We will generate a code book as follows keeping in mind we must practice linked list implementation

Create an array CodeBook of size 128 (0 to 127) where each element of the array is a pointer to a Header node. A Header node will be the same as any other node, it will just be used in a slightly different way. The purpose of the header node in this implementation, is to keep track of how many nodes are in the list. The other nodes in the list contain an Integer data field and a next field. Similar to lecture. Generate 2000 random characters in the range 0 to 127, in other words generate random integers in the range 0 to 127 where each represents an ASCII character. For example, if you generate a 32, this corresponds to index 32 of the CodeBook.

For each character generated, sequentially number this character and enter it into the linked list corresponding to the character in the CodeBook. As each node is added, a count in the Header node is updated to reflect the length of the list. See below.



For example, if you generate a 0 (null character), this corresponds to the first entry in the code book array (CodeBook index). If that character happened to be the 200th character generated, then a node

containing 200 is entered into the list associated with character 0, see above. If a subsequent character 0 is generated and it happens to be the 761 character you generate, then it is entered into the same list as a node with entry 761. Note, that the list is sorted. Thus, since the characters are generated in sequential increasing order, then adding a new node to the end of a list will maintain a sorted order.

Once you have generated the CodeBook, print it out by enumerating each index in the CodeBook to generate the table below. This will confirm to you, that you can generate a CodeBook and that your linked lists are not broken. The first column of the table is the ASCII code of the character. Some characters are invisible or can't be printed, so printing their corresponding integer equivalent is fine. This is just the CodeBook array index.

0	200	761	1100	
1	19			
2	79	401		
3				
4				
5				

When creating the CodeBook, create the array of Node pointers and initialize each pointer with a Header node, where the integer field is 0, indicating the list is empty.

Write a method:

```
private void Add(Node List, int aCode){
        code to add a new node to the end of the list,  List.next. Note List is the header of the
        list so we add at List.next.
}
```

which will be called as follows:

```
Add(CodeBook[aChar], aCode)
```

CodeBook[aChar] is a pointer to a List header node, aChar is the integer equivalent of a character; and aCode is the integer code which will be entered into the list.

For the purpose of this assignment we will keep the CodeBook in memory, thus for the next 2 sections we will encode and decode in quick succession avoiding having to store the code book to a file.

Encoding

You are given an ASCII text file **Message.txt** which you are to encrypt. Read each character aChar, in the text file 1 at a time. That character will be an index into the CodeBook, e.g. CodeBook[(int) aChar]. Emit a random integer from the list associated with aChar. The Header node contains a count of how many nodes are in the list. Generate a random integer RandInt in the range from 1 to Size of list. Write a simple loop that will walk down the list and then emit the code at that node. E.g. from the example CodeBook, if we read a null character (character 0), we generate a random integer in the range 1 to 3, Say we generate a 2. Then count down 2 nodes in the list and emit 761 as the code. Write the emitted

codes to an ASCIIOutput File, **Encrypted.txt**. Note, that when we read a character aChar we cast this to an integer so we can index CodeBook.

Decoding

Read the ASCII File you created above. Read an integer from that file one at a time. Traverse the entire CodeBook until you locate the integer within the CodeBook. Emit the character associated with the list. E.g. Assume your read the integer 401, scanning the CodeBook you will find 401, when the CodeBook index is 2, thus you emit ASCII character 2. Write these emitted characters to a 2nd ASCII output file called **Decrypted.txt**. You can cast the CodeBook index back to a char, e.g. (char) CodeBookIndex.

If you have done everything correctly **Decrypted.txt** should be the same as **Message.txt**.

Some Advice

1. This assignment is best done in stages. Implement the CodeBook and print it out in table format. If you can't get this to go it is pointless continuing.
2. ASCII data files have readC and writeC methods which should be used instead of readChar and writeChar. This is because the latter two methods require or add delimiters, which will cause problems.

Submission:

1. Your Java source code properly commented.
2. CodeBook output, a table showing your Codebook in the form above.
3. Encrypted.txt.
4. Decryped.txt.

Zip all these up and submit via Sakai. Be sure that parts 2 to 4 above are from the same program execution. This is to show continuity, from the CodeBook to the encryption to the decryption.

fin…