CSE 3318 Lab Assignment 4

Due April 17

Goals:

1. Understanding of binary search trees.

2. Understanding of recursive binary tree traversal.

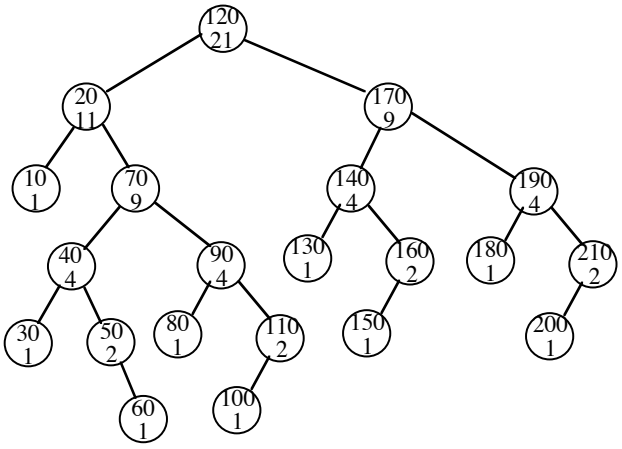
Requirements:

1. Use C to implement 1) serialization/marshalling/unloading/flattening of a binary search tree to a string and 2) the inverse operation of deserializing/unmarshalling/loading/unflattening a string to a binary search tree. Both operations are based on the recursive pre-order traversal of a binary tree.

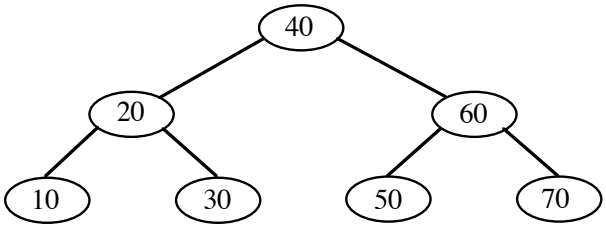
   The input is 1) the number of bytes in a string (including the NULL terminator), 2) a string no longer than the indicated length corresponding to a binary search tree, 3) `n`, the number of keys to be inserted into the tree, and 4) the `n` integers to be inserted into the tree. These will be provided through standard input.

   The output is 1) the length (including the NULL terminator) of a string corresponding to the final binary search tree (after insertions) and 2) the string corresponding to the final binary search tree.

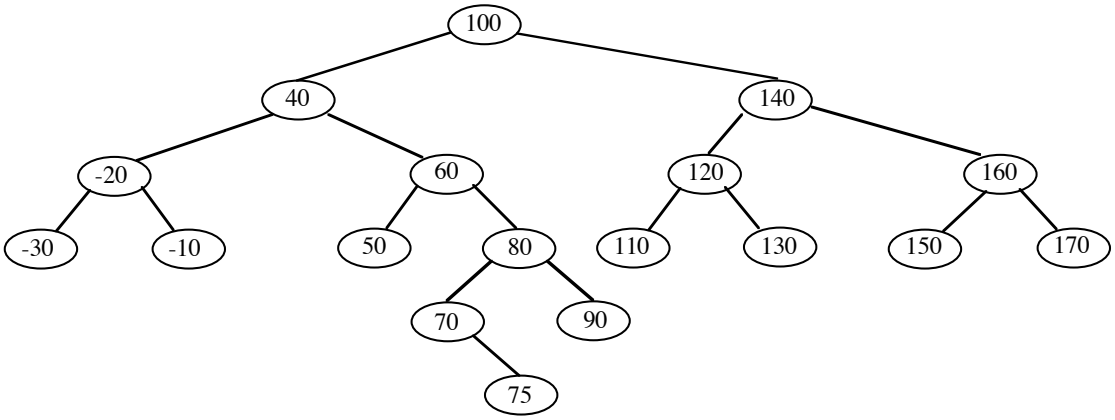
   In the serialized version of a tree, `.` indicates the sentinel. Each key will be preceded by a `+` or `–` indicating its sign. Three examples follow:




```
98 +120+20+10..+70+40+30..+50.+60..+90+80..+110+100...+170+140+130..+160+150...+190+180..+210+200...
```




```
30 +40+20+10..+30..+60+50..+70..
```




```
82 +100+40-20-30..-10..+60+50..+80+70.+75..+90..+140+120+110..+130..+160+150..+170..
```

2. Submit all necessary C source files on Canvas by 3:45 pm on Wednesday, April 17. Comments at the beginning of the source file should include: your name, your ID number, and the command used to compile your code on Omega (5 point penalty for non-compliance).

Getting Started:

1. Suitable driver and header files are available at `http://ranger.uta.edu/~weems/NOTES3318/LAB/LAB4SPR24/` .

2. You must use separate compilation. Do not merge together implementation and header files.

3. The string representing a binary search tree will be free of spaces.

4. Be sure your code does not leak memory. If you `malloc()` it, you are obligated to `free()` it.

5. You should check the deserialized tree either while building it or by using `STverifyProperties()`.

6. Your deserialization code should check the input string for errors. Characters past the end of a tree should result in a warning. Inappropriate characters elsewhere should result in a message and `exit()` termination.

7. `sprintf()` in `stdio.h` will be very useful for string concatenation in your serialization code.

8. Your deserialization code should not use key insertion.

9. It is convenient to have a global character pointer for accessing the string being deserialized.

10. Your deserialization code should set the subtree sizes.