CS:1210 (22C:16) Homework 1

Due via ICON on Thursday, Feb 13th, 4:59 pm

What to submit: Your submission for this homework will consist of three text files, named `hw1a.py`, `hw1b.py`, and `hw1c.py`. These should contain Python programs for Problems (a), (b), and (c) respectively. These files should each start with with a comment block containing your name, section number, and student ID. You will get no credit for this homework if your files are named differently, have a different format (e.g., docx), and if your files are missing your information.

(a) Write a Python program that reads a binary number as input and outputs the equivalent decimal number. For example, if the input is `11001` then the output should be `25`. You should assume that the input consists of a contiguous sequence of 0's and 1's and nothing else. In other words, your program need not consider possible errors in the input.

When your program is run in Wing, the session in the Python shell should look like:

```
[evaluate hw1a.py]
11001
25
```

Note that the program is somewhat minimal in the output it produces; it does not produce a prompt while waiting for input and it does not produce an output message, just the answer. Your program should behave exactly like this because your programs will be automatically graded to a large extent and any other type of behavior, even if it is technically correct, will throw our grading scripts off and you will end up losing points.

(b) A positive integer n is called an *abundant number* if its factors, excluding itself, add up to a quantity greater than n. For example, 12 is an abundant number because its factors excluding itself are 1, 2, 3, 4, and 6 and $1 + 2 + 3 + 4 + 6 = 16$. You can verify that 18 is also an abundant number. Mathematicians interested in number theory have studied abundant numbers; follow the link `http://en.wikipedia.org/wiki/Abundant_number` to take a look at Wikipedia's page on abundant numbers.

Write a Python program that reads a positive integer N and outputs a list of all abundant numbers less than or equal to N. For example, the above mentioned Wikipedia page on abundant numbers lists 12, 18, 20, 24, 30, 36, 40, 42, 48 as all the abundant numbers less than or equal to 50. So your program should produce this list ad output if the input is 50. In other words, when your program is run in Wing, the session in the Python shell should look like:

```
[evaluate hw1b.py]
50
12
18
20
24
30
36
40
42
48
```

The first number in the above list is the input and the sequence of numbers after that is the list of all abundant numbers less than or equal to 50. As is Problem (a) you can assume that there are no errors in the input. In other words, the input is guaranteed to be a positive integer.

(c) *Run length encoding* is very simple form of data compression that is best illustrated with an example. Suppose you want to transmit the string `AAAAABBBBBBBBCC` to a friend. Instead of sending this string as is, you could compress it to `5A8B2C` and send the smaller string. The smaller string will tell your friend that the original string consists of 5 `A`'s followed by 8 `B`'s followed by 2 `C`'s. This type of compression is called run length encoding because *runs* in the data (i.e., contiguous subsequences consising of the same character) can be encoded as a count followed by the data item.

For this problem you are asked write a program that reads input consisting of a string that is run length encoded. Your program is expected to then output the original (i.e., unencoded) string. For example, if the input to your program is

```
5
A
8
B
2
C
0
```

then your program should produce the string `AAAAABBBBBBBBCC` as output. The 0 at the end of your input is being used to indicate that there is no further input. Later, when your Python skills are more developed, you will be able to read `5A8B2C` as a single string and break it up into its constituent parts. For now, you can assume that the input is nicely specified in the format:

```
count1
string1
count2
string2
...
...
countk
stringk
0
```

Corresponding to the above input, your program should produce a string that starts with `count1` copies of `string1`, followed by `count2` copies of `string2`, etc. As in previous problems, you can assume that there will be no errors in the input. Also, as in previous problems, your program should *not* produce a prompt when waiting for input and it should *not* produce any output except the answer.