Problem 4: Transposition Cipher (loop)

A very simple transposition cipher `encrypt`(S) can be described by the following rules:

1. If the length of S is 1 or 2, then `encrypt`(S) is S.
2. If S is a string of N characters $s_1 s_2 s_3 \ldots s_N$ and $k = \lfloor N/2 \rfloor$, then

$$\texttt{enc}(S) = \texttt{encrypt}(s_k s_{k-1} \ldots s_2 s_1) + \texttt{encrypt}(s_N s_{N-1} \ldots s_{k+1})$$

   where + indicates string concatenation.

For example, `encrypt("Ok") = "Ok"` and `encrypt("12345678") = "34127856"`.

Write a program to implement this cipher, given an arbitrary text file input up to 16 MB in size. Start with the template program found at https://6.s096.scripts.mit.edu/grader/text/loop/loop.zip. In this program, you will see a mostly complete function to read a file into a dynamically allocated string as required for this problem.

```
size_t getstr( char **str, FILE *input ) {
  size_t chars_to_read = BLOCK_SIZE;
  size_t length = 0;
  // ...snipped... see template file
  size_t chars = 0;
  while( ( chars = fread( *str + length, 1, chars_to_read, input ) ) ) {
    // you fill this out
  }
  // ...snipped... see template file
  return length;
}
```

Read through the code carefully, make sure you understand it, and complete the inner part of the `while` loop. Look up `realloc` and the `<string.h>` header. If you have any questions about the provided code or don't know why something is structured the way it is, please ask about it on Piazza.

You will also see an empty function "encrypt", which you should fill out.

```
void encrypt( char *string, size_t length ) {
  // you fill this out
}
```

Resource Limits

For this problem you are allotted 3 seconds of runtime and up to 32 MB of RAM.

Input Format

Lines 1…: The whole file (can be any number of lines) should be read in as a string.

Sample Input (file loop.in)

```
Test
early
and often!
```

Output Format

Line 1: One integer: the total number of characters in the string
Lines 2…: The enciphered string.

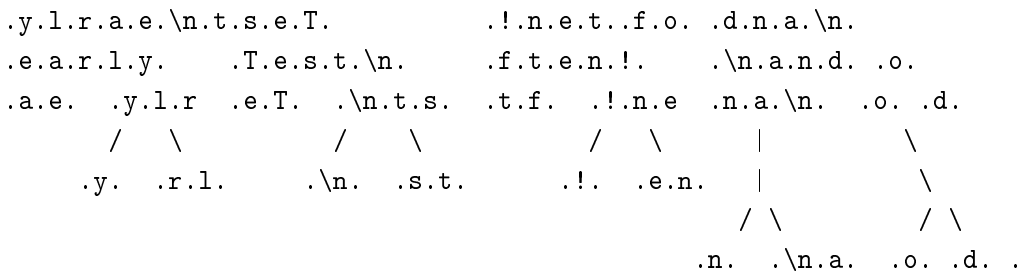
```
21
aeyrleT
sttf!enn
aod
```

Output Explanation

Here's each character in the string as we are supposed to read it in, separated with '.' so we can see the newlines and spaces:

```
.T.e.s.t.\n.e.a.r.l.y.\n.a.n.d. .o.f.t.e.n.!.
```

The string is first split in half and then each half reversed, and the function called recursively; you can see the recursion going on here:



This diagram makes it look a bit more complicated than it actually is. You can see that the sample is correct by reading off the leaves of the tree from left to right–it's the enciphered string we want.

```
.a.e.y.r.l.e.T.\n.s.t.t.f.!.e.n.n.\n.a.o.d. .
```