Set Specs for RLE - status: ready.

Read characters in as uint8_t for both compress and decompress.

Each 8-bit is represented as a uint8_t in code. The compressed format is sentinel followed by a stream of bytes terminated by EOF. If a byte is the **sentinel**, it is **followed by two bytes**: (count) + (character/data). The **count** byte tells the decompressor how many times to expand/repeat the c**haracter/data** byte.

Compress:

1. Read the whole file. Find the character with the **least appearance, or 0 appearance**. **Use that as the sentinel/escape character.**
2. If characters **are NOT repeated** 4+ times (A, AA, AAA)
   They will just be printed out as they are (A, AA, AAA)
3. If a character is **repeated 4+ times**, it will be printed out like this:
   (%)(count)(character)
   Example:
   Input: 55555
   Output: %55

```
uint8_t sentinel = 37;  // which is %
uint8_t count = 5;
uint8_t character = 53; // '5'
printf("%c%c%c", sentinel,count, character);
```

4. If characters are **repeated more than 255 times**:

- Stop reading at 255 and treat the next part as a new 'run'.
  Example:
  Input: "AAAAA….." (A repeated 256 times) so 255 A and another A
  Output: %(255)AA
  Input: "BBBBB……" (B repeated 300 times) so 255 B and 45 more B
  Output: %(255)B%(45)B

5. **Special case** (the escape/sentinel character, % for example)
   If there is a **sentinel in the data**, we'll **compress it no matter if it's repeated or not**.
   So:
   (%)(**count**)(%)
   Input: AAAAA% (6 characters) **// compressing 1 escape character**
   Output: %5A%1% (8 characters)
   Input: AAAAA%%%% (9 characters) **// compressing 4 escape characters**
   Output: %5A%4% (8 characters)

Depress, I mean decompress:

1. Set the first character as the sentinel.
2. If you see characters, just start printing them.
3. If you see the sentinel(% in this case), expect this structure:
   (%)(count)(character)
   And print it out as (character) repeated (count) times.
4. If you see **%(count)%**, print the escape character (count) times.

Example of a decompression process for clarity:
Input: &AAA&AB&BA&1&C&A&

I'll break this down kind of step by step:

1. **&** is the first character of the input, hence, it is the escape character for this file.
2. **AAA** When we read A into uint8_t, it's value will be 65 and not 38(&) hence it's not an escape character and we just print it out. Same for the next 2 A's. -> AAA
3. **&AB**. Next is & with value 38 == 38 (the value for the escape character our program is looking for) hence we will read the next value A 65 as a count, store that, and we then read B as 66, but now we treat that as a character to print. And we will print B 65 times. -> BBBBBBB….(65)
4. **&BA,** this one is the same with the previous one, print A 66 times. -> AAA..(66)
5. **&(1)&,** here, we print the escape character once. -> &
   Note: the 1 in ascii is SOH and, visually, it is not represented as anything.
   For your convenience: https://www.rapidtables.com/code/text/ascii-table.html
6. **C** and then we print the C once. -> C
7. **&A&** and then we print the escape character 65 times. &&&... (65)

IMPORTANT NOTE:

In (sentinel)(count)(character), the count will **not** visually be a number but whatever the ascii value for that number is. So your output might look super weird, but as long as when you decompress it and get the same result as your input for compressing. You have done it correctly.

Input: BBBBB…. (B repeated 45 times)
Output, visually: %-B
Understand it as: %(45)B

When the computer is reading that example (%-B), it will get:

```
uint8_t sentinel   = 37;    // the ascii decimal value for %
uint8_t count      = 45;    // the ascii decimal value for -
uint8_t character = 66;     // the ascii decimal value for B
```

This can visually make you think you have compressed your data incorrectly if you open your compressed file and see something like %5 instead of %55 for an input of “55555” because 5 in ascii is not something you can see...

5	05	00000101	&#5;	ENQ	Enquiry

Hope that makes sense…
Oh here is an interesting example: **%BB should be decompressed to B repeated 66 times**.
Go here to see what your output might look like:
https://www.rapidtables.com/code/text/ascii-table.html

Read the rest **if** you want to know **why** you are doing what you are doing.

Overview on RLE:

Here are some notable stuff from **wiki** (for anyone who is seeing RLE for the first time like me):
Run-length encoding (RLE) is a way to **compress** data without losing any details (**lossless**).

***runs* of data** (sequences in which the same data value occurs in many consecutive data elements) are stored as a single data value and count, rather than as the original run.

Bad for normal texts, good for things like simple graphic images such as icons, line drawings, and animation, …

Example: B- Black W- White (black and white image/icon maybe)

> **Input**:
> WWWWWWWWWWWWWBWWWWWWWWWWWWWWBBBWWWWWWWWWWWWWWWW
> WWWWWWWWWWWBWWWWWWWWWWWWWW (67 characters)
>
> **Compressed**: (we are **not** supposed to do it this way.)
> 12W1B12W3B24W1B14W (18 characters)

RLE boils down to having 2 parts in a compressed file:

1. Character part
2. Number part

Keep this in mind as you go on.

From D2L specifications:

- We are **NOT** to do **RLE in its simplest form (**the previous wiki example**)** since it can **increase the file size.**
  Example: ABC -> 1A1B1C (3 to 6 characters) // so that's a problem.

  Solution:
  Use a **sentinel (**to prevent increasing file size**)**
  Only compress 4+ repeated characters by:
  Printing: **(sentinel)(count)(character)**

  Example: with % being the sentinel
  ABC -> ABC (3 to 3)
  AAAABBBBCCCC -> %4A%4B%4C (12 to 9)
  ABBBAAB *AAAA* BBB *CCCC* -> ABBBAAB%4ABBB%4C (18 to 16)

About the **sentinel**:

It's just a character that we choose to use as an escape character. We can pick any.

Dynamic sentinel:

On the **compress side**, we will **read in the whole file** (don't need to worry about the size, it is confirmed that big files will not be used to test the programs). Then **count** and pick the **character that appeared** the **least or 0 times** as the **escape character**.
Then **send** it **as the first character** in the compressed file.

On the **decompress side**, just **read the first character** and, there you go,
that's your sentinel/escape character.

Reason: **it's just better** that way. I'm too lazy to type out why right now, but that's the protocol and that's how we will do it.

Option:

A team could always encode using the same escape character (example: %)
And as long as they **send it as the first character of the file.** It's all good.

But we do not have that option for decompressing.
The sentinel could be different for every compressed file sent in by a different team. So for our decompression process, we have to **read the first char of the input file and set that as the escape character for that particular decompression run.**

Problems:

1. **Sentinels in the data**:

What if there is a sentinel in the data?

Example, say our sentinel is **%** and we need to compress this:
"Yooooo, I think I lost 40**%!!!!!**"

Proposed solution:

Compress every character that is our sentinel. So:
Y%(5)o, I think I lost 40%(1)%%(5)!" (31 to 29 characters)

2. **Numerical data**:

Problem with having a sentinel + number + character
Example:
555558 -> %558 ?? yeah you can see the problem right?

Solution:

The computer can actually tell that the first 5 is a number and the second 5 is a character if we do this:

```
uint8_t count = 5;
printf("%c", count);       // Use this way to print out the count  5
uint8_t character = 53;
printf("%c", character);   // Use this way to print out the character '5'
```

When we read these into uint8_t, the top one will still be uint8_t 5, and the second one will be uint8_t 53, which is '5'

So you can imagine the output would look something like this: %5'5''8' to the computer when it's actually just %558 to us, visually.

D’Arcy clarifications

Use this section to keep the set mutually informed of decisions, responses from D’Arcy.
If you add something here help everyone out: *notify the set on discord that a clarification has been added*

Question: File size?
D’Arcy’s response: Small enough so you can read all at once.

Question: Sentinel choice?
D’Arcy’s response: Preferably dynamically chosen for every file

Question:
D’Arcy’s response:

Test cases

Test Case Tracker