TeLeVision

Problem Description

We are going to write a seeking algorithm that takes in the file `data.bin`, which contains a specialized array that has been serialized. The algorithm will read the array according to its rules. The flag can be printed out by printing the first, middle, and 8 characters of the string made using the below constraints.

You have been provided `data.bin` which contains the serialized array. The array is a TLV (Tag-Length-Value) array, where each array contains a Tag, Length, and Value. Each element takes the following form:

```
| T | L |  Value  |
| 0 | 1 | 2 | ... |
```

The parts of the array entry are:

- *Tag*: An identifier used to classify the type of data that is incoming.
- *Length*: The length of the value.
- *Value*: The value of the data.

You should store all the entries of the array that have prime number tags in a string. No delimiters should be used in between entries.

The flag is generated by printing the first, middle, and 8 characters of the string created. *Hint: The string has an even length.*

The flag will be represented in ASCII. Remember to surround your data in `flag{}` braces.

Walkthrough

You'll notice that this writeup won't define a `struct`. In this case, we don't really need to. Because we only need to store the value of the items that satisfy a condition (being prime), we can just store the values we find.

Reading the Data

For this challenge, we need to read the data as normal:

1. Open the file and ensure it's properly opened
2. Initialize the container
3. Read the data using an infinite while loop

First, we open the file. Nothing new here.

```
FILE* fp = fopen("data.bin", "rb");
if (fp == NULL) {
    printf("File not found.\n");
```

```
    exit(EXIT_FAILURE);
}
```

Then, we initialize the container. The instructions said to store the entries in a string, so I made a large string. It's infeasible to predict the number of elements for two reasons:

1. The size of each ArrayItem is based on `length`
2. We don't know which of the tags are prime

I chose an arbitrarily large number (being 10000). The file is only 8.6 kilobytes, so this number is more than large enough.

```
char* string = malloc(10000 * sizeof(char));
```

Now, we read the data. Since we don't know the number of elements to read, we use an infinite while loop. Rather than tracking the number of elements read, we track the index we are at in the string so we know where to write for each iteration.

When we read the data, we must check if a number is prime at the time of reading. If the number is prime, we should copy it from the variable into the string, starting at `index`.

```
while (1) {
    uint8_t tag;
    uint8_t length;
    char* value;

    uint8_t bytes = fread(&tag, sizeof(tag), 1, fp);
    if (bytes == 0) break;

    fread(&length, sizeof(length), 1, fp);

    value = malloc(length * sizeof(char));
    fread(value, length, 1, fp);

    if (isPrime(tag)) {
        for (uint8_t i = 0; i < length; ++i)
            string[index + i] = value[i];
        index += length;
    }
    free(value);
}
```

Defining the `isPrime` function is simple. We simply can check if the number is divisible by any number between 2 and the number itself. If you want to optimize this further, you only need to check between 2 and `sqrt(n)`. You'll need the math package for this.

```

```

```
int isPrime(uint8_t tag)
{
    if (tag < 2) return 0;

    for (uint8_t i = 2; i <= tag / 2; ++i)
        if (tag % i == 0) return 0;

    return 1;
}
```

Surprisingly, this is all the processing we need to do for this section. We can move on to printing the flag.

Printing the Flag

Printing the flag is a bit difficult. We need the first, middle, and last 8 characters of the string. We're provided the hint that the string has an even length, so we can use this to our advantage. Let's think about the format of a small string:

```
0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
25
a  b  c  d  e  f  g  h  i  j  k  l  m  n  o  p  q  r  s  t  u  v  w  x  y
z
|----- First 8 ------|     |----- Middle 8 -----|    |------ Final 8 ------
|
```

How can we print these out?

- For the first 8, we can iterate starting at index `0` and go `8` elements.
- For the last 8, we can count backwards `8` from the end of the string and start there.
- For the middle 8, we need to find the middle of the string, move `8/2` to the left, and then iterate `8` elements.

Here's how we can make this work. I chose to use a variable for the characters per section (`cps`) to have more expandable code.

```
uint8_t cps = 8;

printf("flag{");
for (uint8_t i = 0; i < cps; ++i)
    printf("%c", string[i]);
for (uint8_t i = 0; i <= 7; ++i)
    printf("%c", string[strlen(string)/2 - cps/2 + i]);
for (int16_t i = cps; i > 0; --i)
    printf("%c", string[strlen(string) - i]);
printf("}\n");
```

Running this gets the flag:

```
$ gcc -o solve solve.c && ./solve && rm solve
flag{&dGe04H<H&jNBSzT1h/7jf;h}
```

Full Solution

Here is the full solution:

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>

int isPrime(uint8_t tag)
{
    if (tag < 2) return 0;

    for (uint8_t i = 2; i <= tag / 2; ++i)
        if (tag % i == 0) return 0;

    return 1;
}

int main()
{
    FILE* fp = fopen("data.bin", "rb");
    if (fp == NULL) {
        printf("Error opening file\n");
        exit(EXIT_FAILURE);
    }

    char* string = malloc(10000 * sizeof(char));

    uint32_t index = 0;
    while (1) {
        uint8_t tag;
        uint8_t length;
        char* value;

        uint8_t bytes = fread(&tag, sizeof(tag), 1, fp);
        if (bytes == 0) break;

        fread(&length, sizeof(length), 1, fp);

        value = malloc(length * sizeof(char));
        fread(value, length, 1, fp);

        if (isPrime(tag)) {
            for (uint8_t i = 0; i < length; ++i)
                string[index + i] = value[i];
            index += length;
```

```
        }
        free(value);
    }

    uint8_t cps = 8;

    printf("flag{");
    for (uint8_t i = 0; i < cps; ++i)
        printf("%c", string[i]);
    for (uint8_t i = 0; i <= 7; ++i)
        printf("%c", string[strlen(string)/2 - cps/2 + i]);
    for (int16_t i = cps; i > 0; --i)
        printf("%c", string[strlen(string) - i]);
    printf("}\n");

    return 0;
}
```