bitio.py

Start by downloading `bitio.py` from `assignment2_files.tar.gz` on eClass.

1 Reading Bits

Part 1: One Bit At A Time
Create a file `simple.txt` with a short message (e.g. `hello` or `algorithm`). Your task is to read this file one **bit** at a time using `bitio.py`. Create a Python script that does the following.

- Open the file `simple.txt` with mode `'rb'`. The `b` means we will read bytes, not characters.
- Instantiate an instance of `BitReader` from `bitio.py`, using the file you just opened as the argument to the constructor.
- Now repeatedly call the `readbit` method and print the returned `0` or `1` to the screen. It may be helpful if you add a space to your output after every 8 bits.
- Notice that an exception is raised when you try to read after the last bit in the file. Make your Python script handle this exception so it does not crash after you read the last bit.

Part 2: Multiple Bits At A Time
Modify your script from last part so it reads 8 bits at a time using the `readbits` method with `n=8`. Print these values along with the character they represent, one per line.

Recall you can use `chr(x)` to get the character corresponding to the integer representing its ASCII code, e.g. `chr(121)` will be `'y'`.

2 Writing Bits

Part 1: Writing Bits
Write a Python script to write the following bits to a file called `message.txt`.

```
01110011011101000111010101100100011110010010
00000110100001100001011100100110100
```

In particular, have your script do the following.

- Open the file `message.txt` with mode `'wb'` to write a byte object to a file (it is ok if the file does not exist yet).
- Instantiate an instance of `BitWriter` from `bitio.py`, using the file you just opened as the argument to the constructor.
- Write these bits one bit at a time using the `writebit` method. It might help if you copy/paste the bit string (from the file on eClass associated with this lecture) into your Python script as a string (add quotes), iterate through the string, and call the `writebit` method with `True` for a 1-bit and `False` for a 0-bit.

Run you script! Can you read the message in the file now?

Part 2: Writing Partial Bytes
The following is the ASCII code for `hello`. The spaces are added for convenience.

```
01101000 01100101 01101100 01101100 01101111
```

Now consider the string that is one bit shorter.

```
01101000 01100101 01101100 01101100 0110111
```

The last bit is missing. Do the same as above with this sequence of bits (i.e. with the last bit missing). Look at `out.txt`. Can you figure out what happened? Look at the implementation of BitWriter!

Part 3: Writing Multiple Bytes
Finally, repeat the task yet again except this time use the following sequence of integers (copy/paste to your code in a Python list).

```
87, 101, 32, 97, 114, 101, 32, 116, 104, 101, 32, 66, 111, 114, 103, 46,
10, 82, 101, 115, 105, 115, 116, 97, 110, 99, 101, 32, 105, 115, 32,
102, 117, 116, 105, 108, 101, 33, 10
```

Regard each as an 8-bit integer and write to the output file using the `writebits` method with `n=8`.

Part 4: Flushing
A common source of confusion with the current assignment is how `BitWriter` flushes the last byte when it is finished. Do the following tasks to get a better understanding of this.

- Read the `flush` and `__del__` methods of `BitWriter`. Also read `writebit`. When can a call to `flush` occur?
- Add a print statement to the `__del__` method printing whatever you would like. Now try running your solution to Part 3 above. In this same method, try printing the value of `self.accumulator`.
- Check that your thoughts on the solution to the first bullet point agrees with what you saw in the previous bullet point.

I suggest you explicitly call `flush()` at the end of any routine where you use the `BitWriter` and expect the output file to be completed at the end of that routine.