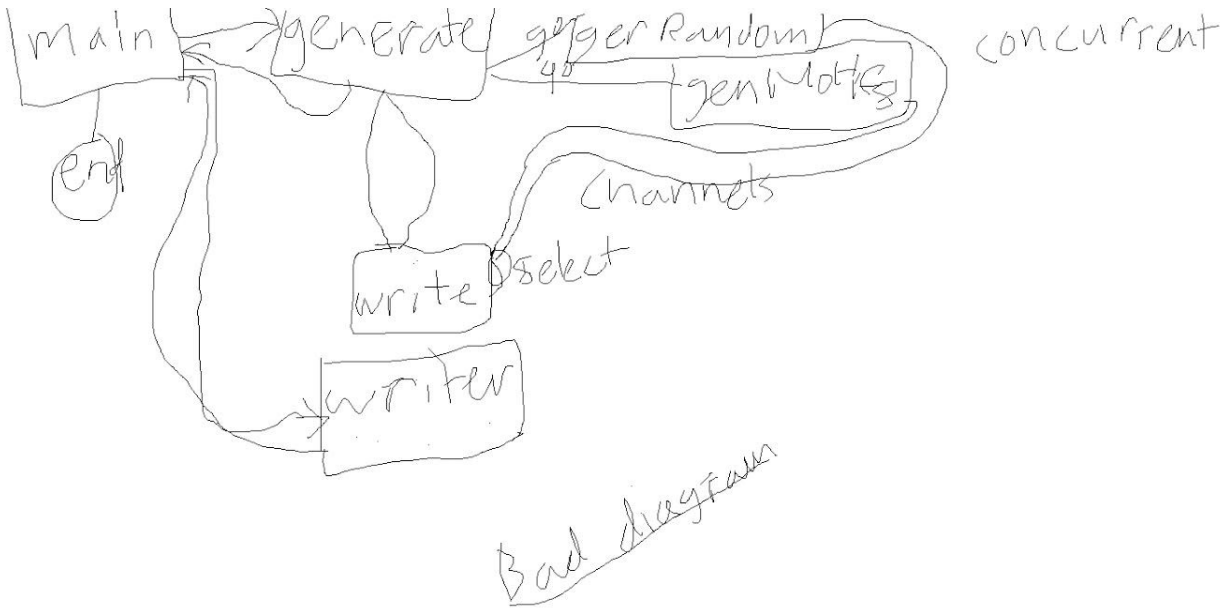
Compilation:
The project is completely contained in main.go, so simply “go build addr/main.go” will compile the project. However, as per the assignment instruction, I did use the flags for the parameters, rather than prompting the user for each parameter. The flags are detailed below, with instructions and an example. The program will run fine without flags: the default values are based on the recommendations in the assignment prompt.

1. **-length=100** the **length** tag defines the length of each sequence. It should equal an integer.
2. **-quantity=50** the **quantity** tag defines the number of sequences. It should equal an integer.
3. **-sizes=“10, 20, 20, 30”** the **sizes** tag defines the lengths of the motifs (and the number of motifs, indirectly). It must equal a **string**, which contains **a list of integers, separated by a comma and a space**.
4. **-mutations= “5, 4, 5, 4”** the **mutations** tag defines the number of mutations which occur on each motif. There must be an equal number of mutations as sizes. Like the sizes tag, this tag takes a string of comma and space separated integers.
5. **-minimum=2** the **minimum** tag defines the minimum number of motifs which occur within each sequence. It should equal an integer.
6. **-subregion=100** the **subregion** tag defines the size of the region in the sequence in which the motifs must appear

Specifications:
The program uses 2 goroutines concurrently generated mutated motifs and random sequences to fill the subregions. The rest of the sequence is randomly filled, and then stored in a global array, which is later written to the file by writer.

Below is my diagram



As far as summarizing the design, the program takes the optional values from the flags, ending and printing an error if the parameters are illegal, and then uses the parameters to generate the templates. The program then calls generate, which starts two concurrent goroutines, each sending to a channel, one which randomly generates sequences of random size, up to 1/10 the size of the subregion, and one

which passes mutated motifs. The generate function, meanwhile, loops through and call the write function for each desired sequence. The write function random chooses the subregion, and fills it using the 2 channels. The program goes through, filling in the region, and at the end checks to ensure the minimum number of motifs have been inserted. If not, it simply goes back it does the same process again until the minimum is met. The rest of the sequence is then filled by a separate function, fill. Then, the entire sequence is converted to a string, the header is added, and this sequence is stored globally. Once all of these sequences are generated, the generator sends a shutdown signal to the 2 concurrent goroutines, and then return control to main. Main calls writer, which outputs the sequences and motifs from their global storage locations. Thus, the files are written and program ends.