Reading Files with a Buffer

This lesson explains how to use a buffer for reading purposes in Go.

We combine the techniques of buffered reading of a file and command-line flag parsing in the following example. If there are no arguments, what is typed in is shown again in the output. Arguments are treated as filenames, and when the files exist, their content is shown. Test it out with:

```
cat test.txt
```

Environment Variables

Key:	Value:
GOROOT	/usr/local/go
GOPATH	//root/usr/local/go/src
PATH	//root/usr/local/go/src/bin:/usr/local/go...

```
Welcome to Educative
```

Click the **RUN** button, and wait for the terminal to start. Type `cat test.txt` and press ENTER.

The `main()` function starts at **line 21**. We parse command-line flags at **line 22**. At **line 23**, we test the case that there are no flags. Then, we make a new buffered reader that will take input from the keyboard, and call the `cat` function on that reader at **line 24**. Starting a for-loop at **line 26** over all the flag arguments, we open the file with the name given in the i-th argument at **line 27**. Error-handling is done from **line 29** to **line 33**. In case of an error, we print out an error message (from **line 30** to **line 31**) and continue with the next flag.

Then, at **line 34**, we construct a buffered reader on that file and call the `cat` function on this reader. Look at the header of the `cat` function at **line 10**. It

takes a pointer to a buffered reader as an argument. In an infinite for-loop (see implementation from **line 11** to **line 17**), it reads all bytes from the buffer until a newline **\n** is encountered. When the end of the file is reached, we return from the function to `main()` (from **line 13** to **line 15**). Then, at **line 16**, we print out what was read.

Apart from buffer, Go also provides functionality of reading and writing files with slices. Move on to the next lesson to see how this works.