Reading and Writing Files with Slices

This lesson explains in detail how to use a slice for reading and writing to files.

Slices provide the standard-Go way to handle I/O buffers; they are used in the following second version of the function `cat`, which reads a file in an infinite for-loop (until end-of-file EOF) in a sliced buffer, and writes it to standard output.

```
func cat(f *os.File) {
  const NBUF = 512
  var buf [NBUF]byte
  for {
    switch nr, err := f.Read(buf[:]); true {
    case nr < 0:
      fmt.Fprintf(os.Stderr, "cat: error reading: %s\n", err.Error())
      os.Exit(1)
    case nr == 0: // EOF
      return
    case nr > 0:
      if nw, ew := os.Stdout.Write(buf[0:nr]); nw != nr {
        fmt.Fprintf(os.Stderr, "cat: error writing: %s\n")
      }
    }
  }
}
```

Look at the following example:

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 `go run main.go test.txt` and press ENTER.

This program is mostly the same as the one covered in the previous lesson. The `main()` follows the same structure, but instead of calling `cat` on a buffered reader to a file, we call it on the file itself; we refer to the explanation there.

If the i^{th} command-line argument is a file `f`, we call `cat` on that file at **line 37**. The `cat` is defined from **line** 8 to **line 24**. It takes a pointer to a file as an argument. In an infinite for-loop (from **line 11** to **line 23**), it reads `NBUF` bytes from the file into a new byte slice `buf` at **line 12**.

Here, `nr, err := f.Read(buf[:]);` is an initialization statement, and `true` simply ensures that the switch will execute. Whereas, `nr` is the number of bytes read. When `nr` is negative, a problem has occurred, and we stop the program with `os.Exit(1)` at **line 15**. When `nr` is 0, it means we are at the end of the file, and we return to `main()` (see **line 17**). In the case that we read some bytes (see **line 18**), we show the buffer on display. The number of bytes written to display is `nw`. **Line 19** tests that if the number of bytes read is not the same as the number of bytes written, there is a problem. Then we display an error message at **line 20**.

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.

Now that you're familiar with the file reading and file writing, let's learn how to scan inputs in the next lesson.