A Practical Example of Interfaces

This lesson provides an explanation of the use of interfaces for writing to the console by using a buffer.

The following program is a nice illustration of the concept of interfaces in `io`:

```
package main
import (
"bufio"
"fmt"
"os"
)

func main() {
  // unbuffered: os.Stdout implements io.Writer
  fmt.Fprintf(os.Stdout, "%s\n", "hello world! - unbuffered")
  // buffered:
  buf := bufio.NewWriter(os.Stdout)
  // and now so does buf:
  fmt.Fprintf(buf, "%s\n", "hello world! - buffered")
  buf.Flush()
}
```

Interfaces in io Package

Here is the actual signature of `fmt.Fprintf()`:

```
func Fprintf(w io.Writer, format string, a ...interface {}) (n int, err er
ror)
```

It doesn't write to a file, it writes to a variable of type `io.Writer`, that is: `Writer` defined in the `io` package:

```
type Writer interface {
  Write(p []byte) (n int, err error)
}
```

The function `fmt.Fprintf()` writes a string according to the format string of

its first argument, which must implement `io.Writer`. `Fprintf()` can write to

any type that has a `Write` method, including `os.Stdout`, files (like `os.File`), pipes, network connections, channels, and buffers from the `bufio` package.

This package defines a type `Writer struct { ... }`. The `bufio.Writer` implements the `Write` method:

```
func (b *Writer) Write(p []byte) (nn int, err error)
```

It also has a factory: give it an `io.Writer`, it will return a buffered `io.Writer` in the form of a `bufio.Writer`:

```
func NewWriter(wr io.Writer) (b *Writer)
```

Buffering works for anything that writes.

> **Remark**: Never forget to use `Flush()` when terminating buffered writing, else the last output won't be written.

To transmit a data structure across a network or to store it in a file, it must be encoded and then decoded again. Many encodings exist, e.g., JSON, XML, gob, Google's protocol buffers, and others. Go has implementations for all of them; in the next *three* sections, we will discuss them.