Lua - File I/O

I/O library is used for reading and manipulating files in Lua. There are two kinds of file operations in Lua namely implicit file descriptors and explicit file descriptors.

For the following examples, we will use a sample file test.lua as shown below.

```
-- sample test.lua
-- sample2 test.lua
```

A simple file open operation uses the following statement.

```
file = io.open (filename [, mode])
```

The various file modes are listed in the following table.

Sr.No.	Mode & Description
1	**"r"** Read-only mode and is the default mode where an existing file is opened.
2	**"w"** Write enabled mode that overwrites the existing file or creates a new file.
3	**"a"** Append mode that opens an existing file or creates a new file for appending.
4	**"r+"** Read and write mode for an existing file.
5	**"w+"** All existing data is removed if file exists or new file is created with read write permissions.
6	**"a+"** Append mode with read mode enabled that opens an existing file or creates a new file.

Implicit File Descriptors

Implicit file descriptors use the standard input/ output modes or using a single input and single output file. A sample of using implicit file descriptors is shown below.

```
-- Opens a file in read
file = io.open("test.lua", "r")

-- sets the default input file as test.lua
io.input(file)

-- prints the first line of the file
print(io.read())

-- closes the open file
io.close(file)

-- Opens a file in append mode
file = io.open("test.lua", "a")

-- sets the default output file as test.lua
io.output(file)

-- appends a word test to the last line of the file
io.write("-- End of the test.lua file")

-- closes the open file
io.close(file)
```

When you run the program, you will get an output of the first line of test.lua file. For our program, we got the following output.

```
-- Sample test.lua
```

This was the first line of the statement in test.lua file for us. Also the line "-- End of the test.lua file" would be appended to the last line of the test.lua code.

In the above example, you can see how the implicit descriptors work with file system using the io."x" methods. The above example uses io.read() without the optional parameter. The optional parameter can be any of the following.

Sr.No.	Mode & Description
1	**"*n"** Reads from the current file position and returns a number if exists at the file position or returns nil.
2	**"*a"** Returns all the contents of file from the current file position.
3	**"*l"** Reads the line from the current file position, and moves file position to next line.
4	**number** Reads number of bytes specified in the function.

Other common I/O methods includes,

- **io.tmpfile()** − Returns a temporary file for reading and writing that will be removed once the program quits.
- **io.type(file)** − Returns whether file, closed file or nil based on the input file.
- **io.flush()** − Clears the default output buffer.
- **io.lines(optional file name)** − Provides a generic *for* loop iterator that loops through the file and closes the file in the end, in case the file name is provided or the default file is used and not closed in the end of the loop.

Explicit File Descriptors

We often use explicit file descriptor which allows us to manipulate multiple files at a time. These functions are quite similar to implicit file descriptors. Here, we use file:function_name instead of io.function_name. The following example of the file version of the same implicit file descriptors example is shown below.

```
-- Opens a file in read mode
file = io.open("test.lua", "r")

-- prints the first line of the file
print(file:read())

-- closes the opened file
file:close()

-- Opens a file in append mode
file = io.open("test.lua", "a")
```

```
-- appends a word test to the last line of the file
file:write("--test")

-- closes the open file
file:close()
```

When you run the program, you will get a similar output as the implicit descriptors example.

```
-- Sample test.lua
```

All the modes of file open and params for read for external descriptors is same as implicit file descriptors.

Other common file methods includes,

- **file:seek(optional whence, optional offset)** − Whence parameter is "set", "cur" or "end". Sets the new file pointer with the updated file position from the beginning of the file. The offsets are zero-based in this function. The offset is measured from the beginning of the file if the first argument is "set"; from the current position in the file if it's "cur"; or from the end of the file if it's "end". The default argument values are "cur" and 0, so the current file position can be obtained by calling this function without arguments.
- **file:flush()** − Clears the default output buffer.
- **io.lines(optional file name)** − Provides a generic *for* loop iterator that loops through the file and closes the file in the end, in case the file name is provided or the default file is used and not closed in the end of the loop.

An example to use the seek method is shown below. It offsets the cursor from the 25 positions prior to the end of file. The read function prints remainder of the file from seek position.

```
-- Opens a file in read
file = io.open("test.lua", "r")

file:seek("end",-25)
print(file:read("*a"))

-- closes the opened file
file:close()
```

You will get some output similar to the following.

```
sample2 test.lua
--test
```

You can play around all the different modes and parameters to know the full ability of the Lua file operations.