COMPUTER PROGRAMMING FILE I/O

Though it's simple to handle file I/O in computer programming, it's really difficult to teach what are the files and how we perform input and output into those files, especially when I promised in pre-requisite of the tutorial that you do not need to know anything about computer.

OK, let's start with learning what are files in computer terminology?

Computer Files

A computer file is used to store data in digital format like plain text, image data or any other content. If you will have a look at this HTML file, name is **computer_programming_file_io.htm** and it's keeping HTML text, which you are reading. Similar way, we use digital image data in the form of files. This tutorial is using minilogo image with a file name **cp-mini-logo.png**. Computer files can be organized inside different directories. So, files are used to keep digital data where as directories are used to keep files.

Computer files can be considered as the digital counterpart of paper documents, which traditionally are kept in office. While doing programming, you keep your source code in text files with different extensions, for example, C programming files have **.c** extension, Java programming files have **.java** extension and Python programming file have extension as **.py**.

File Input/Output

Usually, you create files using text editors like notepad, MS Word, MS Excel or MS Powerpoint, etc., but many times, we need to create files using computer program as well. We can modify existing file using a computer program.

File input means data, which we write into a file and file output means data, which we read from a file. Actually, input and output terms are more related to screen input and output where we display our result on the screen which is called output and if we provide some input to our program from command prompt, then it's called input.

For now, it's enough to remember that writing into a file is file input and reading something from the file is file output.

File Operation Modes

Before we start playing with any file using our program, either we need to create a new file if it does not exist or open an already existing file. In either case, we can open a file in the following modes:

- **Read Only Mode** : If you are going just to read an existing file and you do not want to write any further content in the file, then you will open file in read only mode. Almost, all the programming languages provide syntax to open file in read only mode.
- **Write Only Mode** : If you are going to write into either an existing file or newly created file but you do not want to read any written content in the file, then you will open file in write only mode. All the programming languages provide syntax to open file in write only mode.
- **Read & Write Mode** : If you are going to read as well as write into the same file, then you will open file in read & write mode. Almost, all the programming languages provide syntax to open file in read & write mode.
- **Append Mode** : When you open a file for writing, it allows you to start writing your content from the beginning of the file but writing content from the beginning in a file, which already has some content, will overwrite already existing content. We prefer to open a file in such a way that we should start appending content in already existing content of the file. So in such situation, we open file in append mode. Append mode is ultimately a write mode, which allows content to be appended in the last of the file. Almost, all the programming languages provide syntax to open file in append mode.

Now, following section will teach you how to open a fresh new file, how to write content in that file and later how to read and append more content into the same file.

Opening Files

You can use the **fopen** function to create a new file or to open an existing file, this call will initialize an object of the type **FILE**, which contains all the information necessary to control the stream. Following is the prototype, i.e., signature of this function call:

```
FILE *fopen( const char * filename, const char * mode );
```

Here, **filename** is string literal, which you will use to name your file and access **mode** can have one of the following values:

Mode	Description
r	Opens an existing text file for reading purpose.
w	Opens a text file for writing, if it does not exist then a new file is created. Here, your program will start writing content from the beginning of the file.
a	Opens a text file for writing in appending mode, if it does not exist then a new file is created. Here, your program will start appending content in the existing file content.
r+	Opens a text file for reading and writing both.
w+	Opens a text file for reading and writing both. It first truncate the file to zero length if it exists otherwise create the file if it does not exist.
a+	Opens a text file for reading and writing both. It creates the file if it does not exist. The reading will start from the beginning but writing can only be appended.

Closing a File

To close a file, use the **fclose** function. The prototype i.e. signature of this function is:

```
 int fclose( FILE *fp );
```

The **fclose** function returns zero on success, or **EOF**, special character, if there is an error in closing the file. This function actually flushes any data still pending in the buffer to the file, closes the file, and releases any memory used for the file. The EOF is a constant defined in the header file **stdio.h**.

There are various functions provided by C standard library to read and write a file character by character or in the form of a fixed length string. Let us see few of them in the next section.

Writing a File

Following is the simplest function to write individual characters to a stream:

```
int fputc( int c, FILE *fp );
```

The function **fputc** writes the character value of the argument c to the output stream referenced by fp. It returns the written character written on success, otherwise **EOF** if there is an error. You can use the following functions to write a null-terminated string to a stream:

```
int fputs( const char *s, FILE *fp );
```

The function **fputs** writes the string **s** into the file referenced by fp. It returns a non-negative value on success, otherwise **EOF** is returned in case of any error. You can use **int fprintf** $FILE * fp, const char * format, ...$ function as well to write a string into a file. Try the following example:

```
#include <stdio.h>

main()
{
   FILE *fp;

   fp = fopen("/tmp/test.txt", "w+");
   fprintf(fp, "This is testing for fprintf...\n");
   fputs("This is testing for fputs...\n", fp);
   fclose(fp);
}
```

When the above code is compiled and executed, it creates a new file **test.txt** in **/tmp** directory and writes two lines using two different functions. Let us read this file in next section.

Reading a File

Following is the simplest function to read a text file character by character:

```
int fgetc( FILE * fp );
```

The **fgetc** function reads a character from the input file referenced by fp. The return value is the character read, or in case of any error it returns **EOF**. The following functions allow you to read a string from a stream:

```
char *fgets( char *buf, int n, FILE *fp );
```

The functions **fgets** reads up to n - 1 characters from the input stream referenced by fp. It copies the read string into the buffer **buf**, appending a **null** character to terminate the string.

If this function encounters a newline character '\n' or the end of the file EOF before they have read the maximum number of characters, then it returns only the characters read up to that point including new line character. You can also use **int fscanf***FILE * fp, constchar * format, . . .* function to read strings from a file but it stops reading after the first space character encounters.

```
#include <stdio.h>

main()
{
   FILE *fp;
   char buff[255];

   fp = fopen("/tmp/test.txt", "r");
   fscanf(fp, "%s", buff);
   printf("1 : %s\n", buff );

   fgets(buff, 255, (FILE*)fp);
   printf("2: %s\n", buff );

   fgets(buff, 255, (FILE*)fp);
   printf("3: %s\n", buff );
   fclose(fp);

}
```

When the above code is compiled and executed, it reads the file created in previous section and produces the following result:

```
1 : This
2: is testing for fprintf...

3: This is testing for fputs...
```

Let's see a little more detail about what happened here. First **fscanf** method read just **This** because after that it encountered a space, second call is for **fgets**, which reads the remaining line

till it encountered end of line. Finally, last call **fgets** reads second line completely.

File I/O in Java

Java programming language provides even richer set of functions to handle File I/O. For a complete detail, I will suggest you to check Java Tutorial.

Here, we will see a simple Java program, which is equal to C program explained above. This program will open a text file and will write few text lines into that file and close the file. Finally, same file is opened and then read that text from already created file. You can try to execute following program to see the output:

```
import java.io.*;

public class DemoJava
{
   public static void main(String []args) throws IOException
   {

      File file = new File("/tmp/java.txt");

      // Create a File
      file.createNewFile();

      //  Creates a FileWriter Object using file object
      FileWriter writer = new FileWriter(file);

      // Writes the content to the file
      writer.write("This is testing for Java write...\n");
      writer.write("This is second line...\n");

      // Flush the memory and close the file
      writer.flush();
      writer.close();

      // Creates a FileReader Object
      FileReader reader = new FileReader(file);
      char [] a = new char[100];

      // Read file content in the array
      reader.read(a);
      System.out.println( a );

      // Close the file
      reader.close();
   }
}
```

When above program is executed, it produces the following result:

```
This is testing for Java write...
This is second line...
```

File I/O in Python

Following program shows the same functionality to open new file, write some content into the file and finally read the same file:

```
# Create a new file
fo = open("/tmp/python.txt", "w")

# Writes the content to the file
fo.write( "This is testing for Python write...\n");
fo.write( "This is second line...\n");

# Close the file
fo.close()
```

```
# Open existing file
fo = open("/tmp/python.txt", "r")

# Read file content in a variable
str = fo.read(100);
print str

# Close opened file
fo.close()
```

When the above code is executed, it produces the following result:

```
This is testing for Python write...
This is second line...
```