Input and Output with Files

C also lets us read and write data to files using fopen() and fclose().

We'll cover the following

- Opening and Closing files with fopen() and fclose()
- Reading and Writing to files
    - Ascii Files (plain text)

Opening and Closing files with `fopen()` and `fclose()`

Before a file can be read or written to, it has to be **opened** using the `fopen()` function, which takes as arguments a string corresponding to the filename, and a second argument (also a string) corresponding to the **mode**. The mode is read ("r"), write ("w") or append ("a"). The `fopen()` function then returns a pointer to the (open) file. After reading and/or writing to your file, you will need to **close** it using the `fclose()` function.

Reading and Writing to files

There are many functions in `stdio.h` for reading from and writing to files. There is a collection of functions for reading and writing ascii (text) data, and there are functions for dealing with binary data.

Ascii Files (plain text)

There are functions to read single characters at a time (`getc()` and `putc()`), there are functions to read and write formatted output (`fscanf()` and `fprintf()`), and there are functions to read and write single lines at a time (`fgets()` and `fputs()`).

Here is an example program that outputs a table of temperature values in Fahrenheit and Celsius to an ascii file.

```
#include <stdio.h>
```

```
int main(int argc, char *argv[]) {

  FILE *fp;

  double tmpC[11] = {-10.0, -8.0, -6.0,
                     -4.0, -2.0,  0.0,  2.0,
                      4.0,  6.0,  8.0, 10.0};
  double tmpF;
  int i;

  fp = fopen("output/outfile.txt", "w");
  if (fp == NULL) {
    printf("sorry can't open outfile.txt\n");
    return 1;
  }
  else {
    // print a table header
    fprintf(fp, "%10s %10s\n", "Celsius", "Fahrenheit");
    for (i=0; i<11; i++) {
      tmpF = ((tmpC[i] * (9.0/5.0)) + 32.0);
      fprintf(fp, "%10.2f %10.2f\n", tmpC[i], tmpF);
    }
    fclose(fp);
  }

  return 0;
}
```

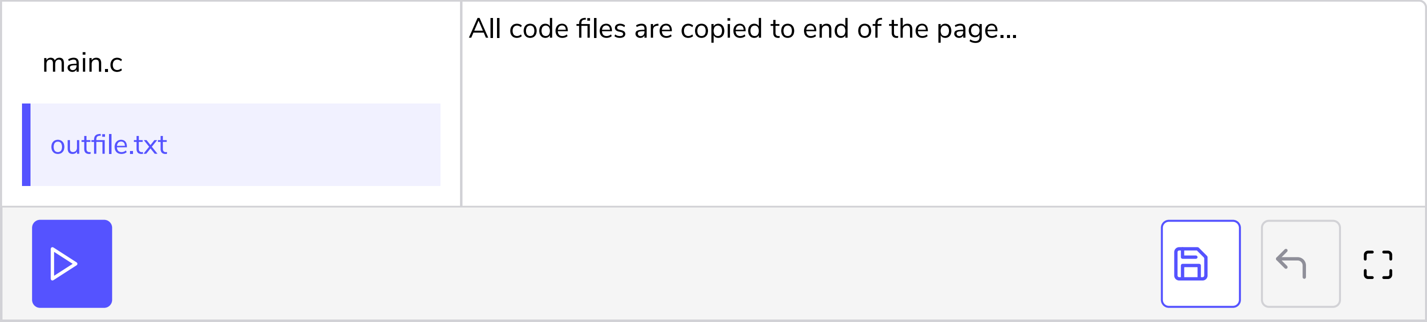
```
plg@wildebeest:~/Desktop$ more outfile.txt
   Celsius Fahrenheit
    -10.00      14.00
     -8.00      17.60
     -6.00      21.20
     -4.00      24.80
     -2.00      28.40
      0.00      32.00
      2.00      35.60
      4.00      39.20
      6.00      42.80
      8.00      46.40
     10.00      50.00
```

A couple of things are worth noting about the code above. On line 13, we check the value of the file pointer `fp`, and if it is equal to `NULL` (which means there was an error opening the file), we write a message to the screen and we `return 1` (which exits the `main()` function and thus exits our program). A convention in UNIX is that programs which execute successfully return `0` and non-zero values are returned when there was an error encountered.

On lines 19 and 22 we use the `fprintf()` function to write to the file. This is just like the `printf()` function that we have seen before, to write formatted output to standard output. This time we're writing to a file instead.

standard output. This time we're writing to a file instead.

To illustrate reading from ascii files, here's an example program that will read in the file produced by the previous code example, and do some arithmetic on them.



All code files are copied to end of the page…

```
tempC=-10.00, tempF=14.00
tempC=-8.00, tempF=17.60
tempC=-6.00, tempF=21.20
tempC=-4.00, tempF=24.80
tempC=-2.00, tempF=28.40
tempC=0.00, tempF=32.00
tempC=2.00, tempF=35.60
tempC=4.00, tempF=39.20
tempC=6.00, tempF=42.80
tempC=8.00, tempF=46.40
tempC=10.00, tempF=50.00
11 values read, sumC=0.00 and sumF=352.00
```

Some comments about the above code example: on line 19 we use the `fgets()` function to read in the first line of the file to a character string (`buffer`) that we declared above. The `fgets()` function requires as its second argument the maximum number of characters to read. Since we know we don't expect many here, we indicate a maximum of 256. After reading in the first line, we now enter a **while loop**, using `fscanf()` to read in each pair of floating-point values. The while loop terminates when `!feof(fp)` is false. The `feof()` function returns TRUE if we are at the end of the file, and FALSE otherwise.

We've seen how C interacts with files in order to read or write text. C extends the same functionality to binary files. We'll look into that next.

Code Files Content !!!

⇓ ⇓

⇓

```
-------------------------------------------------------------------------
|  main.c [1]
-------------------------------------------------------------------------


#include

int main(int argc, char *argv[]) {

  FILE *fp;
  char buffer[256];
  double tempC, tempF;
  double sumC = 0.0;
  double sumF = 0.0;
  int numread = 0;

  fp = fopen("outfile.txt", "r");
  if (fp == NULL) {
    printf("there was an error opening outfile.txt\n");
    return 1;
  }
  else {
    // read in the header line first
    fgets(buffer, 256, fp);
    while (!feof(fp)) {
      fscanf(fp, "%lf %lf\n", &tempC, &tempF);
      printf("tempC=%.2f, tempF=%.2f\n", tempC, tempF);
      sumC += tempC;
      sumF += tempF;
      numread++;
    }
    fclose(fp);
    printf("%d values read, sumC=%.2f and sumF=%.2f\n", numread, sumC, sumF);
  }

  return 0;
}



-------------------------------------------------------------------------
|  outfile.txt [1]
-------------------------------------------------------------------------


   Celsius Fahrenheit
    -10.00      14.00
     -8.00      17.60
     -6.00      21.20
     -4.00      24.80
     -2.00      28.40
      0.00      32.00
      2.00      35.60
      4.00      39.20
      6.00      42.80
      8.00      46.40
     10.00      50.00
```

```
**********************************************************************************
```