File Streams

Section 5.1

Introduction

- A **stream** is a flow of water
- Analogy used in programming for a stream of bytes
  - Bytes flowing from one source to another
    - Files, users, devices, etc.

FILE

- **FILE** is a C data type for representing a file
  - Actually a structure defined in **stdio.h**
- When want to work with a file, make a pointer to a **FILE**
  - **FILE *fptr;**

fopen()

- To open a file, we use **fopen()**
  - When opening the file succeeds, **fopen()** returns a pointer to the file
  - When opening fails, returns **NULL**
    - Good to check to make sure opening successful

- **fopen()** takes two inputs
  - Name of the file (as a string)
  - Access mode (as a string)
- Access mode is a letter(s) describing what want to do with the file

- Several different access modes, common ones are:
  - **"r"** – read the file (want to access, but not modify it)
    - File must already exist
  - **"w"** – write to the file (creates the file if it does not already exist, overwrites it if it does)
  - **"a"** – append to the file (add on to an existing file, creates new file if doesn't already exist)

- Some others as well:
  - **"r+"** – read, but also allowed to write (modify)
  - **"w+"** – makes an empty file that can both be read and written to (overwrites already existing file of same name, if exists)
  - **"a+"** – opens file for both reading and writing, writing is appended to end of file

```
FILE *new_file;
new_file = fopen("data.txt", "w");
```

```
FILE *file;
file = fopen("book.txt", "r");
```

fclose()

- **fclose()** used to close a file
  - Need to close a file when done working with it
  - Input is a FILE pointer for an open file

```
fclose(new_file);
```

```
fclose(file);
```

fprintf()

- **fprintf()** used for writing text to a file
- Works like printf(), but specifies a file to write the text to, rather than printing the text on the screen
- First parameter is a pointer to the file
- Other parameters are like for printf()

```
FILE *new_file;

float money = 19.99999;

new_file = fopen("data.txt", "w");

fprintf(new_file, "hello, how are you?\n");

fprintf(new_file, "Money: %.2f\n", money);

fclose(new_file);
```

fscanf()

- To read text from a file, we can use **fscanf()**
- Works like scanf(), but reads from a file, rather than user input
- First parameter is a FILE pointer
- Other parameters are like for scanf()

```
FILE *file;

char text[100];

file = fopen("book.txt", "r");

fscanf(file, "%s", text);

fclose(file);
```

EOF

- **EOF** stands for end of file
- Special value used to indicate when the end of a file is reached
  - Often is simply the value of -1
- Useful for knowing when to stop a loop

- **fscanf()** (as well as **scanf()**) return an int indicating how many items they successfully read
- When they reach the end of a file, they return EOF
- Useful for processing a file line by line using a loop

```
FILE *file;
int a;
int b;

file = fopen("data.txt", "r");

while (fscanf(file, "%d %d\n", &a, &b) != EOF)
{
    printf("Values are: %d and %d\n", a, b);
}

fclose(file);
```

fgetc()

- **fgetc()** can be used to read a single character from a text file
  - Input is simply the FILE pointer
  - Returns the character read or EOF if at end of file
    - Note that it returns an int, not a char
- Often used when want to process a file character by character

```
FILE *file;
int c;

file = fopen("data.txt", "r");

while ((c = fgetc(file)) != EOF)
{
    printf("c: %c\n", c);
}

fclose(file);
```

fread()

- fscanf() assumes we're reading from a text file
- **fread()** is a more basic and generic way of reading from a file
- Basically tell it how many bytes to read from the file
- FILE structure keeps track of current position in file (how many bytes read so far)

- **fread()** has four parameters
  - First is where to read the bytes to
  - Second is units of bytes (1 byte? 4 bytes? etc.)
  - Third is how many units
  - Fourth is pointer to file to read from

```
FILE *file;

char text[80];

file = fopen("book.txt", "r");

fread(text, 1, 15, file);

text[15] = '\0';

printf("%s\n", text);

fclose(file);
```

fwrite()

- fprintf() assumes writing text to a file
- **fwrite()** more basic and generic way to write data to file
- Like with **fread()**, tell it how many bytes to write

```
FILE *file;
int array[10];
int i;
for (i = 0; i < 10; i++)
    array[i] = 5;

file = fopen("file.txt", "w");

fwrite(array, sizeof(int), 10, file);

fclose(file);
```

Standard Streams

- Standard streams are streams that exist by default
  - Actually FILE pointers
- Have special and common uses, set up automatically
  - **stdin** (Standard In)
  - **stdout** (Standard Out)
  - **stderr** (Standard Error)

stdin

- **stdin** represents standard input
- stdin is what we're reading from when we read input from the user
- **scanf()** is like a version of fscanf(), but specifically for stdin

stdout

- **stdout** represents standard output
- Where stuff goes when we print it
- **printf()** is like a specialized version of fprintf() that specifically prints to the stdout, which is usually the console

stderr

- **stderr** stands for standard error
- A place we can send error information too
- A way to separate regular output from error information
- Typically also goes to the screen like stdout

System IO Functions

- IO functions with f-prefix are from Standard C Library
  - Same behavior regardless of system
- Also have OS dependent IO functions
  - **open()**, **close()**, **read()**, and **write()**
  - Can vary by system
  - Cover in Chapter 7