Each of these modes can end with a + character. This means that the file is to be opened for both reading and writing.

Mode	Meaning
"r+"	open text file for reading and writing
"w+"	open text file for writing and reading
.....	

Opening for reading a file that cannot be read, or does not exist, will fail. In this case fopen() returns a NULL pointer. Opening a file for writing causes the file to be created if it does not exist and causes it to be overwritten if it does. Opening a file in append mode causes the file to be created if it does not exist and causes writing to occur at the end of the file if it does.

A file is opened for updating (both reading and writing) by using a + in the mode. However, between a read and a write or a write and a read there must be an intervening call to fflush() to flush the buffer, or a call to one of the file positioning function calls fseek(), fsetpos(), or rewind().

In some operating systems, including UNIX, there is no distinction between binary and text files, except in their contents. The file mechanism is the same for both types of files. In MS-DOS and other operating systems, there are different file mechanisms for each of the two types of files. (See exercise 22, on page 548, for further discussion.)

A detailed description of file handling functions such as fopen() and fclose() can be found in Section A.12, “Input/Output: <stdio.h>,” on page 655. Consult the appendix as necessary to understand how the various functions are used.

11.5 An Example: Double Spacing a File

Let us illustrate the use of some file handling functions by writing a program to double-space a file. In main(), we open files for reading and writing that are passed as command line arguments. After the files have been opened, we invoke double_space() to accomplish the task of double spacing.

In file dbl_space.c

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

void   double_space(FILE *, FILE *);
void   prn_info(char *);
```

```
int main(int argc, char **argv)
{
  FILE   *ifp, *ofp;

  if (argc != 3) {
    prn_info(argv[0]);
    exit(1);
  }
  ifp = fopen(argv[1], "r");     /* open for reading */
  ofp = fopen(argv[2], "w");     /* open for writing */
  double_space(ifp, ofp);
  fclose(ifp);
  fclose(ofp);
  return 0;
}

void double_space(FILE *ifp, FILE *ofp)
{
  int  c;

  while ((c = getc(ifp)) != EOF) {
    putc(c, ofp);
    if (c == '\n')
     putc('\n', ofp);   /* found a newline – duplicate it */
  }
}

void prn_info(char *pgm_name)
{
  printf("\n%s%s%s\n\n%s%s\n\n",
    "Usage:  ", pgm_name, "  infile  outfile",
    "The contents of infile will be double–spaced ",
    "and written to outfile.");
}
```

Suppose we have compiled this program and put the executable code in the file *dbl_space*. When we give the command

dbl_space file1 file2

the program will read from *file1* and write to *file2*. The contents of *file2* will be the same as *file1*, except that every newline character will have been duplicated.