【C】Day6

Course	Advanced C
Study Date	@April 12, 2022

【Ch7】Input and Output

7.6 Error Handling-Stderr and Exit

The current error handling is not ideal. If one of the files cannot be accessed for some reason, the diagnostic is printed at the end of the concatenated output.

That output might go into a file or another program via a pipeline.

To handle error better, a second output stream, called `stderr`, is assigned to a program in the same way that `stdin` and `stdout` are.

Output written on `stderr` normally appears on the screen even if the standard output is redirected.

```
#include <stdio.h>

int main(int argc, char* argv[]) {
  FILE *fp;
  void file_copy(FILE*, FILE*);
  char *prog = argv[0];

  if(argc == 1)
    file_copy(stdin, stdout);
  else
    while(--argc > 0) {
      if((fp = fopen(*++argv, "r")) == NULL) {
        fprintf(stderr, "%s failed to open file %s\n", prog, *argv);
        exit(1);
      } else {
        file_copy(fp, stdout);
        fclose(fp);
      }
    }

  if(ferror(stdout)) {
    fprintf(stderr, "%s: error  writing stdout\n", prog);
    exit(2);
  }
```

```
  return 0;
}
```

The program signals errors two ways:

1. First, the diagnostic output produced by `fprintf` goes onto `stderr`, so it finds its way to the screen instead of disappearing down a pipeline or into an output file.
2. Second, the program uses the standard library function `exit`, which terminates program execution when it is called.

   The argument of exit is available to whatever process called this one, so the success or failure of the program can be tested by another program that uses this one as a sub-process.

Conventionally, a return value of 0 signals that all is well; non-zero values usually signal abnormal situations.

`exit` calls `fclose` for each open output file, to flush out any buffered output.

Within main, `return expr` is equivalent to `exit(expr)`. exit has the advantage that it can be called from other functions.

The function `ferror` returns non-zero if an error occurred on the stream fp.

```
int ferror(FILE *fp)
```

The function `feof(FILE*)` is analogous to `ferror`. It returns non-zero if end of file has occurred on the specified file.

```
int feof(FILE *fp)
```