Fully Commented & Clarified Code

```
/*
 * A simple program to read a file and write its contents to stdout.
 *
 * Steps to write this code:
 * 1. Figure out which syscalls/wrappers we need: open(), read(), write(),
close().
 *    - `man 2 open`  → shows open() prototype, flags, return values.
 *    - `man 2 read`  → how to read bytes from a file descriptor.
 *    - `man 2 write` → how to write bytes to a file descriptor.
 *    - `man 2 close` → how to close file descriptors.
 * 2. Decide on headers:
 *    - <fcntl.h>   → for O_RDONLY and other file open flags.
 *    - <stdio.h>   → for printf() (used in error messages).
 *    - <unistd.h>  → for read(), write(), close(), and access to syscall
wrappers.
 *    - <stdlib.h>  → for exit() and EXIT_* macros.
 * 3. Choose buffer size → define COUNT = 100 bytes per read().
 * 4. Compile with: `gcc -o mycat mycat.c`
 * 5. Run with: `./mycat myfile.txt`
 */



#include <fcntl.h>    // for open() flags like O_RDONLY
#include <stdio.h>    // for printf() to print errors/help
#include <unistd.h>   // for read(), write(), close()
#include <stdlib.h>   // for exit()

#define COUNT 100     // buffer size for each read()

int main(int argc, char *argv[]) {
    char buf[COUNT];  // buffer to hold file data temporarily

    // Step 1: check command-line arguments
    // Program should be run as: ./mycat filename
    if (argc != 2) {
        printf("Usage: %s file-name\n", argv[0]);
        exit(-1);
    }

    // Step 2: open the file (read-only mode)
```

```c
    // man 2 open → int open(const char *pathname, int flags);
    int fd = open(argv[1], O_RDONLY);
    if (fd < 0) {  // open() returns -1 on error
        printf("Could not open the file\n");
        exit(-2);
    }

    // Step 3: read() and write() loop
    // - read() attempts to fill buf with up to COUNT bytes
    // - returns number of bytes read, or 0 if EOF, or -1 on error
    int num_read;
    while ((num_read = read(fd, buf, COUNT)) > 0) {
        // write() to stdout (fd = 1)
        if (write(1, buf, num_read) < 0) {
            printf("Write failed\n");
            exit(-3);
        }
    }

    // Step 4: close file descriptor when done
    close(fd);

    return 0;
}
```

🔹 Notes on Building This Code

1. Finding Prototypes

- Run `man 2 open` → shows `int open(const char *pathname, int flags, mode_t mode);`
  - Needs `<fcntl.h>` for the flags (`O_RDONLY`, `O_WRONLY`, etc.).
- Run `man 2 read` → `ssize_t read(int fd, void *buf, size_t count);`
  - Declared in `<unistd.h>`.
- Run `man 2 write` → `ssize_t write(int fd, const void *buf, size_t count);`
  - Also in `<unistd.h>`.
- Run `man 2 close` → `int close(int fd);`
  - Also in `<unistd.h>`.

2. Headers

- `<unistd.h>` → provides the syscall wrappers (`read` , `write` , `close`).
- `<fcntl.h>` → provides open flags like `O_RDONLY` .
- `<stdio.h>` → used here for `printf()` (error messages).
- `<stdlib.h>` → for `exit()` .

3. Compilation

```
gcc -Wall -Wextra -o mycat mycat.c
```

- `-Wall -Wextra` → enable warnings (good practice).
- `-o filecopy` → output binary named `filecopy` .

4. Running

```
./mycat myfile.txt
```

- Opens `myfile.txt` , reads it in chunks of 100 bytes, and writes to `stdout` .
- Works like a very simplified version of the `cat` command.