

# Problem Solving using Files

Rupesh Nasre.

Mentor TAs: Ahmed, Vivek, Vimala, Akash,  
Kankan, Rahul, Swati, Akshay, Ashok, Keshav

IIT Madras  
May 2022

# Files allow persistent data.

- Our programs so far were transient.
  - Once a.out is terminated, the data is lost.
  - In many applications, data entered one day should be available on some other day.
- Files make data persistent.
  - Across multiple invocations of a.out
  - Across multiple executables also (a.out, b.out)
- We can read data stored in files, or write to it.

# File Operations

- **fopen**: opens a file.
- **fclose**: closes it.
- **fscanf**, **fgets**, **fread**: Read data from a file.
- **fprintf**, **fputs**, **fwrite**: Write data to a file.
- These operations work with a file pointer.

# Problem: Implement *cat*.

If the file does not exist, or is not readable, **fopen** returns NULL.

- *cat* allows printing contents of a file.

```
int main(int a, char *b[]) {  
    FILE *fp = fopen(b[1], "r");  
    char str[100];  
  
    fgets(str, 100, fp);  
    while (!feof(fp)) {  
        puts(str);  
        fgets(str, 100, fp);  
    }  
    fclose(fp);  
}
```

File pointer  
automatically  
moves ahead.  
Same as scanf.

Does this miss the last  
line in the file?

```
$ ./a.out mycat.c  
int main(int a, char *b[]) {  
  
    FILE *fp = fopen(b[1], "r");  
  
    char str[100];  
  
    fgets(str, 100, fp);  
  
    ...  
}
```

- `fgets` retains `\n` in `str`. Either remove it prior to `puts`, or print it using `printf` without extra `\n`.
- Recall that `gets` does not include `\n`.
- Both `fgets` and `gets` end the string with `\0`.

# Problem: Implement cp.

If the file does not exist, w mode creates a new file.  
Use "a" to append to a file.

- Copy a file to another.

```
int main(int a, char *b[]) {  
    FILE *fpr = fopen(b[1], "r");  
    FILE *fpw = fopen(b[2], "w");  
    char str[100];  
  
    fgets(str, 100, fpr);  
    while (!feof(fpr)) {  
        fputs(str, fpw);  
        fgets(str, 100, fpr);  
    }  
    fclose(fpr);  
    fclose(fpw);  
}
```

# Problem: Implement cp.

- Copy a file to another: from input to output.

```
int main(int a, char *b[]) {  
    FILE *fpr = fopen(b[1], "r");  
    FILE *fpw = fopen(b[2], "w");  
    char str[100];  
  
    Same as gets fgets(str, 100, stdin);  
    Ctrl-d is eof while (!feof(stdin)) {  
    Same as puts fputs(str, stdout);  
        fgets(str, 100, stdin);  
    }  
    fclose(fpr);  
    fclose(fpw);  
}
```

C provides two standard I/O devices:  
**stdin** and **stdout**.

These are always open (and should not be closed).

A third one is called **stderr**, which is primarily used to print error messages.

# Problem: Split text into words.

## \$ mycat file.in

Lexical analyzer groups a sequence of characters into tokens.

Syntax analyzer checks if these tokens adhere to a grammar syntax.

## \$ mysplit file.in file.out

## \$ mycat file.out

- 1: Lexical
- 2: analyzer
- 3: groups
- 4: a
- 5: sequence
- 6: of
- 7: characters
- 8: into
- 9: tokens.
- 10: Syntax
- 11: analyzer
- 12: checks
- 13: if
- 14: these
- 15: tokens
- 16: adhere
- 17: to
- 18: a
- 19: grammar
- 20: syntax.

```
int main(int a, char *b[]) {
    FILE *fpr = fopen(b[1], "r");
    FILE *fpw = fopen(b[2], "w");
    char str[100];

    fscanf(fpr, "%s", str);
    int lineno = 0;
    while (!feof(fpr)) {
        fprintf(fpw, "%d: %s\n", ++lineno, str);
        fscanf(fpr, "%s", str);
    }
    fclose(fpr);
    fclose(fpw);
}
```

# Problem: Join words into text.

```
$ mycat file.out
```

```
1: Lexical  
2: analyzer  
3: groups  
4: a  
5: sequence  
6: of  
7: characters  
8: into  
9: tokens.  
10: Syntax  
11: analyzer  
12: checks  
13: if  
14: these  
15: tokens  
16: adhere  
17: to  
18: a  
19: grammar  
20: syntax.
```

```
int main(int a, char *b[]) {  
    FILE *fpr = fopen(b[1], "r");  
    FILE *fpw = fopen(b[2], "w");  
    char str[100];  
  
    int lineno;  
    fscanf(fpr, "%d: %s", &lineno, str);  
    while (!feof(fpr)) {  
        fprintf(fpw, "%s ", str);  
        if (strstr(str, ".")) fprintf(fpw, "\n");  
        fscanf(fpr, "%d: %s", &lineno, str);  
    }  
    fclose(fpr);  
    fclose(fpw);  
}
```

```
$ myjoin file.out file.in
```

```
$ mycat file.in
```

Lexical analyzer groups a sequence of characters into tokens.

Syntax analyzer checks if these tokens adhere to a grammar syntax.



# Problem: Text correction

- Replace *mispelt* with *misspelt* in a file.
- Unlike previous problems, this demands reading as well as writing to the same file.
  - fopen supports additional modes.
  - r+: Allows reading and writing. Returns NULL if the file cannot be opened.
  - w+: Allows reading and writing, initially erases the file content.
- After reading, file pointer FP moves ahead.
- Thus, after reading *mispelt*, FP needs to go back.

# Problem: Text correction

```
int main(int a, char *b[]) {  
    FILE *fp = fopen(b[1], "r+");  
    char str[100];  
  
    fscanf(fp, "%s", str);  
    while (!feof(fp)) {  
        if (strcmp(str, "mispelt") == 0) {  
  
        }  
        fscanf(fp, "%s", str);  
    }  
    fclose(fp);  
}
```

From the current position (relative)  
Also supported: SEEK\_SET, SEEK\_END

```
$ mycat file.in
```

If a word is misspelt it will be corrected.

```
$ ./a.out file.in
```

```
$ mycat file.in
```

In general, we can use another file for writing.

r+ allows filesize to grow.  
e.g., if *mispelt* is the last word in the file.

# Problem: Student Records

- Read student records from user.
- Store in a file (RAM --> DISK).
- Retrieve records from the file (DISK --> RAM).

To append to an existing file, we can open it in the append mode "a".

## filestud.h

```
struct student {
    char rollno[9];
    char name[20];
    char hostel[20];
    int room;
};
```

After executing a.out, how many student's records will stud.dat contain?

## filestud.c

```
#include "filestud.h"

void writetofile(struct student *stud) {
    FILE *fp = fopen("stud.dat", "w");
    fprintf(fp, "%s\n%s\n%s\n%d\n",
            stud->rollno, stud->name,
            stud->hostel, stud->room);
    fclose(fp);
}

void readfromfile() {
    FILE *fp = fopen("stud.dat", "r");
    ...
    fclose(fp);
}
```

## filestudmain.c

```
#include "filestud.h"

int main() {
    struct student stud;
    populate(&stud);
    writetofile(&stud);
    populate(&stud);
    writetofile(&stud);

    readfromfile();
}
```

# Application: Ticket Booking System

- N seats: 1 to N, initially unoccupied
- Booking needs name, assigns an unoccupied seat
- Cancellation needs seat number
- Update needs seat number, allows change of name.

	<b>Week</b>	<b>Problems</b>	<b>Tools</b>
✓	0	Solve equations, find weighted sum.	Data types, expressions, assignments
✓	1	Find max, convert marks to grade.	Conditionals, logical expressions
✓	2	Find weighted sum for all students.	Loops
✓	3	Encrypt and decrypt a secret message.	Character arrays
✓	4	Our first game: Tic-tac-toe	2D arrays
✓	5	Making game modular, reuse.	Functions
✓	6	Find Hemachandra/Fibonacci numbers.	Recursion
✓	7	Encrypt and decrypt many messages.	Dynamic memory, pointers
✓	8	Maintain student records.	Aggregate data types
✓	9	Search and sort student records.	Searching and sorting algorithms
✓	A	Reduce memory wastage.	Linked lists
✓	B	Implement token system in banks.	Queues
✓	C	IRCTC-like ticket booking system	File handling
✓	D	Putting it all together	All the above