1. Why do we need to know the output buffer? How many ways to flush the output buffer?

    a. The output buffer is used by C standard library functions (like printf(), fputs(), etc.) to store data temporarily before it is written to an actual output (like a terminal or file). Understanding how the output buffer works is essential for the following reasons:
        i. **Efficiency**: Writing data in chunks (buffering) is much more efficient than writing data byte-by-byte.
        ii. **Control Over Output**: Knowing how to control the buffer allows to decide when and how data is actually written,

Method	**Description**
fflush(stdout)	Forces immediate output to stdout
fclose()	Flushing occurs automatically when closing a file stream.
Newline in printf()	Automatically flushes stdout (line-buffered mode).
setvbuf()	Sets custom buffering behavior.
exit(0);	When a program exits normally, all buffers are flushed automatically.

2. Which library functions may cause overflow in Table 15.1 of Pointers-On-C.pdf? How can we prevent it from occurring?

Function	**Unsafe**	**Safe Alternative**
gets()	gets(buffer)	fgets(buffer, sizeof(buffer), stdin)
scanf()	scanf("%s", buffer)	scanf("%49s", buffer)
fread()	fread(buffer, 1, wrong_size, input)	fread(buffer, 1, sizeof(buffer), input)

printf()	printf(input);	printf("%s", input);

3. Get the output based on the given data
    a. Get the time by command date. Hint, Using the knowledge what we learned in this chapter to redirect.
    b. Put all the output data into the variable buf
    c. Show as below.

input_output_p3.c > main(void)

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#define BUF_LEN 128

int main(void) {
    unsigned int sip = 0, dip = 0;
    int len = 0, sport = 0, dport = 0;
    char buf[BUF_LEN] = {0};


    sip = 0x01010101;
    dip = 0x02020202;
    len = 44;
    sport = 23;
    dport = 54177;

    // Get the current time
    time_t now;             // now is variable which can store sytem time.
    struct tm *tm_info;     // a pointer to struct tm varible

    time(&now);             // store system time to now
    tm_info = localtime(&now);  // convert time_t value to struct tm value,
                                // struct tm is human readble



    // Format the time into a string similar to "Fri Feb 12 02:16:09 EST 2025"
    char time_buf[BUF_LEN];
    strftime(time_buf, BUF_LEN, "%a %b %d %H:%M:%S %Z %Y", tm_info);

    // Format the final output string
    snprintf(buf, BUF_LEN, "%s, IP src = %d.%d.%d.%d, dst = %d.%d.%d.%d, len = %d, TCP sport = %d, dport = %d",
             time_buf,
             (sip >> 24) & 0xFF, (sip >> 16) & 0xFF, (sip >> 8) & 0xFF, sip & 0xFF,
             (dip >> 24) & 0xFF, (dip >> 16) & 0xFF, (dip >> 8) & 0xFF, dip & 0xFF,
             len, sport, dport);

    // Print the formatted output
    printf("\n\n%s\n\n", buf);

    return 0;
}

```

output:

```
Mon Feb 17 16:46:06 Bangladesh Standard Time 2025, IP src = 1.1.1.1, dst = 2.2.2.2, len = 44, TCP sport = 23, dport = 54177

PS C:\Users\Admin\Documents\bdcom_coding_zone>
```

4. There is a file whose name is input.txt; it is calculated that the file length is n. Copy the part n/2-n to file output.txt. As efficient as possible.

input_outputp2.c > main()

```c
#include <stdio.h>
#include <stdlib.h> // for Proper exit Handling, EXIT SUCCESS(0), EXIT FAILURE(1)

int main() {
    FILE *input, *output;
    long file_size, start_pos;

    // Open input file for reading
    input = fopen("input.txt", "rb");
    if (input == NULL) {
        perror("Error opening input.txt");
        return EXIT_FAILURE;
    }

    // Open output file for writing
    output = fopen("output.txt", "wb");
    if (output == NULL) {
        perror("Error opening output.txt");
        fclose(input);
        return EXIT_FAILURE;
    }

    // Find the file size
    fseek(input, 0, SEEK_END);      // move pointer to end of the file
    file_size = ftell(input);       // return current position of the pointer from start

    // Calculate start position (n/2 to n)
    start_pos = file_size / 2;      // caculating middle position and set it to new start
    fseek(input, start_pos, SEEK_SET);  // set the pointer to start_pos

    // Efficiently copy using a buffer
    char buffer[4096];  // 4KB buffer
    size_t bytes_read;  // size of the input after fread() call.

    while ((bytes_read = fread(buffer, 1, sizeof(buffer), input)) > 0) {
        fwrite(buffer, 1, bytes_read, output);
    }

    // Close files
    fclose(input);
    fclose(output);

    printf("Successfully copied second half of input.txt to output.txt\n");

    return EXIT_SUCCESS;
}
```

output:

output.txt

```
asdfghjkla
```

5. List out all options of the fopen that you know and their descriptions

Mode	**Read/write**	**Position Indicator**	**If File Exists**	**If File Doesn't Exist**
"r"	Read only	Beginning of file	Opens normally	Error (NULL is returned)
"r+"	Read & Write	Beginning of file	Opens normally	Error (NULL is returned)
"w"	Write only	Beginning of file	**Truncates file** (erases content)	Creates a new file
"w+"	Read & Write	Beginning of file	**Truncates file** (erases content)	Creates a new file
"a"	Write only	End of file (append mode)	Opens normally (writes at the end)	Creates a new file
"a+"	Read & Write	End of file (append mode)	Opens normally (writes at the end)	Creates a new file