1. Need for File Handling in C

Even though programs can run without file handling, it is essential because:

1. **Persistence/Permanent Storage:** Data processed during program execution (stored in RAM) is lost when the program terminates. File handling allows data to be stored **permanently** on a disk[1].
2. **Handling Large Data:** Files allow programs to handle an amount of data much larger than what can fit into the system's memory (RAM) at once.
3. **Data Sharing:** Files enable data to be easily shared between multiple programs or between different runs of the same program.

---

2 & 8. Program to Copy File Contents (Command Line Arguments)

This program copies the contents of a source file (argument 1) to a destination file (argument 2)[2222].

C

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

// argc = argument count, argv = argument vector (array of strings)
int main(int argc, char *argv[]) {
    FILE *source_file, *dest_file;
    char ch;

    // Check if the correct number of arguments is provided (program_name, source, destination)
    if (argc != 3) {
        printf("Usage: %s <source_file> <destination_file>\n", argv[0]);
        exit(1);
    }

    // Open source file in read mode ("r")
    source_file = fopen(argv[1], "r");
    if (source_file == NULL) {
        printf("Error: Cannot open source file %s.\n", argv[1]);
        exit(1);
    }

    // Open destination file in write mode ("w"). Creates it if it doesn't exist.
    dest_file = fopen(argv[2], "w");
    if (dest_file == NULL) {
        printf("Error: Cannot create destination file %s.\n", argv[2]);
```

```
        fclose(source_file);
        exit(1);
    }

    // Read character by character from source and write to destination
    while ((ch = fgetc(source_file)) != EOF) {
        fputc(ch, dest_file);
    }

    printf("File contents successfully copied from %s to %s.\n", argv[1], argv[2]);

    fclose(source_file);
    fclose(dest_file);
    return 0;
}
```

---

3. Graphics: initgraph() and Concentric Circles

Use of initgraph()

The **initgraph()** function is used to initialize the graphics system[3]. It loads the necessary graphics driver into memory and puts the system into graphics mode.

- **Syntax:** void initgraph(int *graphdriver, int *graphmode, char *pathtodriver);
- **Arguments:** It typically requires a driver, a mode, and the path where the BGI (Borland Graphics Interface) files are located.

Program for 50 Concentric Circles

(This requires the non-standard graphics.h library, common in DOS/Turbo C++ environments.)

C

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

void main() {
    int gd = DETECT, gm; // gd = graphics driver, gm = graphics mode
    int x_center, y_center;
    int max_radius = 50; // We want 50 circles, increasing radius by 1

    // Initialize the graphics mode
    initgraph(&gd, &gm, "C:\\TURBOC3\\BGI"); // Adjust path as needed

    // Find the center of the screen
    x_center = getmaxx() / 2;
    y_center = getmaxy() / 2;
```

```
    // Draw 50 concentric circles
    for (int radius = 1; radius <= max_radius; radius++) {
        setcolor(radius % 15 + 1); // Cycle through colors
        circle(x_center, y_center, radius);
    }

    getch(); // Wait for a key press
    closegraph(); // Close the graphics mode
}
```

---

4. Why Files are Needed, File Handling, and fseek()

Why Files are Needed

Files are needed for **data persistence** (saving data permanently) [4] and for managing datasets that exceed the available **RAM** [5].

File Handling Definition

File Handling in C is the mechanism used to read data from, and write data to, files stored on a secondary storage device (like a hard disk). It involves functions to **open**, **read/write**, and **close** files.

fseek() Function

The **fseek()** function is used to **set the file position indicator** (cursor) to a specified byte offset within an opened file [6]. This allows for **random access** to data in the file.

- **Syntax:** int fseek(FILE *fp, long offset, int reference_position);
- **Parameters:**
    - fp: Pointer to the file stream.
    - offset: Number of bytes to move from the reference position.
    - reference_position (or **SEEK_SET**): Specifies the starting point for the offset:
        - 0 or **SEEK_SET**: Beginning of the file.
        - 1 or **SEEK_CUR**: Current position of the file pointer.
        - 2 or **SEEK_END**: End of the file.

---

5. Graphics Functions Explanation

These functions are part of the non-standard **graphics.h** library [7].

Function	Purpose	Syntax/Example
initgraph()	Initializes the graphics system, loading the driver and setting the mode[8].	initgraph(&driver, &mode, "path");
rectangle()	Draws a rectangle on the screen[9].	rectangle(left, top, right, bottom);
line()	Draws a straight line between two specified points[10].	line(x1, y1, x2, y2);
setbgcolor()	Sets the background color for the graphics screen (takes a color code or constant)[11].	setbgcolor(RED);
outtextxy()	Outputs text string at a specified coordinates (X, Y)[12].	outtextxy(100, 100, "Hello");

---

6 & 10. C Preprocessor and Directives

A **C Preprocessor** is a program that processes the source code **before** it is compiled[13]. It performs operations like file inclusion, macro substitution, and conditional compilation. All preprocessor commands begin with the # symbol.

Categories of Preprocessor Directives[14]

1. **File Inclusion:** To include the content of another file into the current file.
    - **Purpose/Example:** #include <stdio.h> (Includes standard library headers)[15].
2. **Macro Substitution (or Definition):** To define constants or function-like macros that are replaced by the preprocessor.
    - **Purpose/Example:** #define PI 3.14159 (Replaces every instance of PI with 3.14159).
3. **Conditional Compilation:** To compile or skip certain parts of the source code based on specified conditions.

    - **Purpose/Example:** #ifdef DEBUG ... #endif (Compiles code only if DEBUG macro is defined)[16].
4. **Other Directives:** Includes directives for error reporting, line control, and pragma directives.
    - **Purpose/Example:** #undef PI (Undefines a previously defined macro).

---

7. fseek (fp, m, 1), feof (), and ferror ()

Explanation of fseek (fp, m, 1)

This statement sets the file pointer fp to a position that is **m bytes** away from the **current position** (1 or SEEK_CUR)[17].

- If m is positive, it moves m bytes forward.
- If m is negative, it moves m bytes backward.

feof() Function

- **Purpose:** Checks if the **End-Of-File** indicator for the given file stream is set[18].
- **Return:** Returns a non-zero value (TRUE) if the EOF indicator is set; otherwise, returns zero (FALSE).
- **Usage:** Used after a read operation to confirm if the end of the file has been reached.

ferror() Function

- **Purpose:** Checks if the **error indicator** for the given file stream is set[19].
- **Return:** Returns a non-zero value (TRUE) if an error has occurred during a file operation; otherwise, returns zero (FALSE).
- **Usage:** Used to check for read/write errors, disk full errors, etc.

---

9. Program: File Character, Word, and Line Count

C

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

void main() {
   FILE *fp;
   char filename[] = "sample.txt";
   char ch;
   int chars = 0, words = 0, lines = 0;
   int in_word = 0; // 0=outside word, 1=inside word

   fp = fopen(filename, "r");
   if (fp == NULL) {
```

```
        printf("Error opening file: %s\n", filename);
        return;
    }

    while ((ch = fgetc(fp)) != EOF) {
        // 1. Line Count
        if (ch == '\n') {
            lines++;
        }

        // 2. Character Count (Counting all bytes including spaces/newlines)
        chars++;

        // 3. Word Count Logic
        // Check for word separators
        if (ch == ' ' || ch == '\n' || ch == '\t') {
            in_word = 0;
        }
        // If character is not a separator AND we are not currently in a word,
        // then a new word has begun.
        else if (in_word == 0) {
            in_word = 1;
            words++;
        }
    }

    // Adjust line count if the file is not empty and doesn't end with a newline
    if (chars > 0 && strchr("\n", ch) == NULL) {
        lines++;
    }

    printf("File Statistics for %s:\n", filename);
    printf("Characters: %d\n", chars);
    printf("Words: %d\n", words);
    printf("Lines: %d\n", lines);

    fclose(fp);
}
```

---

11. Program: Copy File and Count Blank Spaces

C

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

void main() {
    FILE *source_fp, *dest_fp;
```

```
    char ch;
    int blank_spaces = 0;
    char source_name[] = "input.txt";
    char dest_name[] = "output_with_spaces.txt";

    source_fp = fopen(source_name, "r");
    if (source_fp == NULL) {
        printf("Error: Cannot open source file %s\n", source_name);
        return;
    }

    dest_fp = fopen(dest_name, "w");
    if (dest_fp == NULL) {
        printf("Error: Cannot create destination file %s\n", dest_name);
        fclose(source_fp);
        return;
    }

    // Copy file contents and count spaces
    while ((ch = fgetc(source_fp)) != EOF) {
        fputc(ch, dest_fp); // Write to destination file

        if (ch == ' ') {
            blank_spaces++;
        }
    }

    printf("File copied successfully.\n");
    printf("Total blank spaces counted: %d\n", blank_spaces);

    // Write the count of blank spaces to the *end* of the second file
    fprintf(dest_fp, "\n\n--- Total Blank Spaces: %d ---", blank_spaces);

    fclose(source_fp);
    fclose(dest_fp);
}
```

---

12. getc() and putc() Use and Limitations

getc() and putc() Functions [20]

- **getc(FILE *fp):** Reads the next **character** from the specified file stream and returns it as an int.
- **putc(int character, FILE *fp):** Writes the specified **character** (passed as an int) to the specified file stream.

Feature	Use	Limitation
getc()	Simple, efficient reading of single characters from a file.	Reads only one character at a time. Returns EOF (an integer) on failure/end of file, which must be handled.
putc()	Simple, efficient writing of single characters to a file.	Writes only one character at a time.

Program: Keyboard $\to$ File $\to$ Screen I/O

This program reads data from the keyboard, writes it to a file named INPUT, then reads it back from INPUT and displays it[21].

C

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

void main() {
    FILE *fp;
    char ch;
    char filename[] = "INPUT";

    // 1. Read from keyboard and write to file (INPUT)
    fp = fopen(filename, "w");
    if (fp == NULL) { printf("Error opening file for writing.\n"); return; }

    printf("Enter data (end with # and press Enter):\n");
    while ((ch = getchar()) != '#') {
        putc(ch, fp);
    }
    fclose(fp);
    printf("\nData written to file: %s\n", filename);

    // 2. Read from file (INPUT) and display on screen
    fp = fopen(filename, "r");
    if (fp == NULL) { printf("Error opening file for reading.\n"); return; }

    printf("\n--- Data read from file ---\n");
    while ((ch = getc(fp)) != EOF) {
        putchar(ch); // Display the character on the screen
```

```
    }
    printf("\n-------------------------\n");

    fclose(fp);
}
```