File Handling in C: Input and Output Functions

File handling in C involves creating, reading, writing, and closing files using functions provided by the standard input/output library (`stdio.h`). Below is a brief overview of the main functions used in file handling.

Main File Handling Functions

1. **fopen()**:
   - **Purpose**: Opens a file and returns a file pointer.
   - **Prototype**: `FILE *fopen(const char *filename, const char *mode);`
   - **Modes**:
     - `"r"`: Opens for reading (file must exist).
     - `"w"`: Opens for writing (creates a new file or truncates existing file).
     - `"a"`: Opens for appending (creates a new file if it doesn't exist).
     - `"r+"`: Opens for both reading and writing (file must exist).
     - `"w+"`: Opens for both reading and writing (creates a new file or truncates existing file).
     - `"a+"`: Opens for both reading and writing (creates a new file if it doesn't exist).

   Example:

   ```
   c
   Copy code
   FILE *file = fopen("example.txt", "r");
   if (file == NULL) {
       perror("Error opening file");
       return 1;
   }
   ```

2. **fclose()**:
   - **Purpose**: Closes an opened file.
   - **Prototype**: `int fclose(FILE *stream);`

   Example:

   ```
   c
   Copy code
   fclose(file);
   ```

3. **fgetc()**:
   - **Purpose**: Reads a character from a file.
   - **Prototype**: `int fgetc(FILE *stream);`

   Example:

   ```
   c
   Copy code
   char c = fgetc(file);
   ```

4. **fputc()**:

- **Purpose**: Writes a character to a file.
- **Prototype**: `int fputc(int character, FILE *stream);`

Example:

```
c
Copy code
fputc('A', file);
```

5. **fgets()**:
   - **Purpose**: Reads a string from a file.
   - **Prototype**: `char *fgets(char *str, int n, FILE *stream);`

   Example:

   ```
   c
   Copy code
   char buffer[100];
   fgets(buffer, 100, file);
   ```

6. **fputs()**:
   - **Purpose**: Writes a string to a file.
   - **Prototype**: `int fputs(const char *str, FILE *stream);`

   Example:

   ```
   c
   Copy code
   fputs("Hello, World!", file);
   ```

7. **fprintf()**:
   - **Purpose**: Writes formatted output to a file.
   - **Prototype**: `int fprintf(FILE *stream, const char *format, ...);`

   Example:

   ```
   c
   Copy code
   fprintf(file, "The value is: %d\n", 42);
   ```

8. **fscanf()**:
   - **Purpose**: Reads formatted input from a file.
   - **Prototype**: `int fscanf(FILE *stream, const char *format, ...);`

   Example:

   ```
   c
   Copy code
   int value;
   fscanf(file, "%d", &value);
   ```

Example Program: Writing and Reading a File

Here is a complete example that demonstrates opening a file, writing to it, reading from it, and closing it.

```
c
Copy code
#include <stdio.h>
#include <stdlib.h>

int main() {
    FILE *file;
    char filename[] = "example.txt";
    char writeData[] = "Hello, World!";
    char readData[100];

    // Open the file for writing
    file = fopen(filename, "w");
    if (file == NULL) {
        perror("Error opening file for writing");
        return 1;
    }

    // Write data to the file
    fprintf(file, "%s", writeData);
    fclose(file);  // Close the file after writing

    // Open the file for reading
    file = fopen(filename, "r");
    if (file == NULL) {
        perror("Error opening file for reading");
        return 1;
    }

    // Read data from the file
    if (fgets(readData, sizeof(readData), file) != NULL) {
        // Print the data read from the file
        printf("Data read from file: %s\n", readData);
    } else {
        perror("Error reading data from file");
    }

    fclose(file);  // Close the file after reading

    return 0;
}
```

Summary

- **Opening and Closing Files**: Use `fopen()` to open a file and `fclose()` to close it.
- **Reading and Writing Characters**: Use `fgetc()` to read and `fputc()` to write individual characters.
- **Reading and Writing Strings**: Use `fgets()` to read and `fputs()` to write strings.
- **Formatted Input and Output**: Use `fprintf()` to write and `fscanf()` to read formatted data.
- **Error Handling**: Always check the return value of file operations to ensure they succeed, and handle errors appropriately.

File handling is an essential aspect of programming in C, providing mechanisms for persistent data storage and retrieval.