**Error Handling in C**

Last Updated : 20 Mar, 2025

In C programming, error handling is typically done using functions that handle runtime errors, returning error codes or messages to notify the programmer about the failure or incorrect operation. Since C does not provide built-in exception handling like other high-level languages (e.g., try-catch in Java or Python), error handling relies heavily on function return values, global variables, and system calls.

A lot of C function calls return -1 or NULL or set an in case of an error code as the global variable **errno**, so quick tests on these values are easily done with an instance of ‘if statement’.

**What is errno?**

**errno**is a global variable defined in the **<errno.h>**header file that indicates the error that occurred during a function call in C. When a function fails, the errno variable is automatically set to a specific error code, which helps identify the type of error encountered. Different values of errno correspond to different types of errors, providing useful information for error handling in C programs.

**Below is a list of a few different errno values and their corresponding meaning:**

| **errno value** | **Error** |
| --- | --- |
| **1** | Operation not permitted |
| **2** | No such file or directory |
| **3** | No such process |
| **4** | Interrupted system call |
| **5** | I/O error |
| **6** | No such device or address |
| **7** | The argument list is too long |
| **8** | Exec format error |
| **9** | Bad file number |
| **10** | No child processes |
| **11** | Try again |
| **12** | Out of memory |
| **13** | Permission denied |

**Example**

#include *<errno.h>*

#include *<stdio.h>*

int main() {

*// If a file is opened which does not exist,*

*// then it will be an error and corresponding*

*// errno value will be set*

FILE\* fp;

*// opening a file which does not exist*

fp = fopen("GeeksForGeeks.txt", "r");

printf("Value of errno: %d**\n**", errno);

**return** 0;

}

**Different Methods for Error Handling**

Different methods are used to handle different kinds of errors in C. Some of the commonly used methods are:

**1. perror()**

The [**perror()**](https://www.geeksforgeeks.org/perror-in-c/) function is used to print an error message to the standard error stream (stderr). It helps to display the error string based on the global errno variable, which stores the error code set by system calls and library functions.

**Example**

{...}

FILE\* fp;

*// Try opening a non-existent file, which sets errno*

fp = fopen("GeeksForGeeks.txt", "r");

*// Print the errno value after failed file opening*

printf("Value of errno: %d

", errno);

perror("Message from perror");

{...}

**Output**

Value of errno: 2

Message from perror: No such file or directory

**Explanation**: This code attempts to open a non-existent file using fopen(), which sets the errno variable with an error code. It then prints the value of errno and uses perror() to display a descriptive error message related to the failure, helping the user understand the cause of the error, such as “No such file or directory.”

**2. strerror()**

The**strerror()** function is also used to show the error description. This function returns a pointer to the textual representation of the current errno value.

**Example**

{...}

FILE\* fp;

*// Try opening a non-existent file, setting errno*

fp = fopen("GeeksForGeeks.txt", "r");

*// Print errno value and corresponding error message*

printf("Value of errno: %d

", errno);

printf("The error message is : %s

", strerror(errno));

{...}

**Output**

Value of errno: 2

The error message is : No such file or directory

**Explanation:** The code attempts to open a non-existent file, sets errno on failure, and then prints both the error code and the corresponding error message using strerror().

**3. ferror()**

The [**ferror()**](https://www.geeksforgeeks.org/ferror-in-c/) function is used to check if an error occurred during a file operation. It returns a non-zero value if there was an error during the file operation.

**Example**

{...}

FILE\* file = fopen("nonexistent\_file.txt", "r");

**if** (file == NULL) {

*// Print error if file opening fails*

perror("Error opening file");

*// Return non-zero exit status on error*

**return** 1;

}

int c;

*// Read file character by character*

**while** ((c = fgetc(file)) != EOF);

**if** (ferror(file)) {

*// Handle read error*

printf("An error occurred while reading the file.

");

} **else** {

printf("File read successfully.

");

}

{...}

**Output**

Error opening file: No such file or directory

**Explanation:** This code attempts to open a file and handles potential errors using perror() if the file can’t be opened. It reads the file character by character, checks for errors during reading using ferror(), and prints an appropriate message. The file is closed at the end, and the program exits with a status indicating success or failure.

**4. feof()**

The [**feof()**](https://www.geeksforgeeks.org/eof-and-feof-in-c/)function checks whether the end of a file has been reached during reading operations. It helps to identify when there is no more data to read from the file.

**Example**

#include *<stdio.h>*

int main () {

FILE \*fp = fopen("test.txt","r");

**if** (fp == NULL)

**return** 0;

**do** {

*// Taking input single character at a time*

char c = fgetc(fp);

*// Checking for end of file*

**if** (feof(fp))

**break** ;

printf("%c", c);

}**while**(1);

fclose(fp);

**return**(0);

}

**test.txt**

Welcome to GeeksforGeeks

**Output**

Welcome to GeeksforGeeks

**Explanation:**In this code, **feof()** is used to check if the end of the file (EOF) has been reached while reading the file character by character using fgetc(). If feof(fp) returns true, the while loop breaks, stopping further reading of the file.

**clearerr()**

The [**clearerr()**](https://www.geeksforgeeks.org/clearerr-in-c/) function is used to clear the error and EOF flags for a stream. It allows recovery from errors and allows the stream to be reused for further operations.

**Example**

{...}

FILE\* file = fopen("file.txt", "r");

**if** (file == NULL) {

*// Print error if file can't be opened*

perror("Error opening file");

*// Exit with error code*

**return** 1;

}

**if** (ferror(file)) {

*// Handle file operation error*

printf("Error during file operations.

");

}

*// Clear error indicators for the file stream*

clearerr(file);

{...}

**Output**

Error opening the file: No such file or directory

**Explanation:** The code opens a file in read mode, handles potential errors during the file opening and operations, clears any error indicators with clearerr(), and finally closes the file.

**Exit Status**

C programs use the exit() function to terminate the program and return a status code to the operating system. The C standard specifies two constants: **EXIT\_SUCCESS** and **EXIT\_FAILURE**, that may be passed to exit() to indicate successful or unsuccessful termination, respectively. These are macros defined in <stdlib.h> header file.

**Example**

#include *<errno.h>*

#include *<stdio.h>*

#include *<stdlib.h>*

#include *<string.h>*

int main(){

FILE\* fp;

*// Attempt to open a non-existent file in binary mode*

fp = fopen("filedoesnotexist.txt", "rb");

**if** (fp == NULL) {

printf("Value of errno: %d**\n**", errno);

printf("Error opening the file: %s**\n**", strerror(errno));

perror("Error printed by perror");

*// Exit the program with failure status*

exit(EXIT\_FAILURE);

*// This line will not be printed because of exit()*

printf("I will not be printed**\n**");

}

*// If the file is opened successfully*

**else** {

fclose(fp);

exit(EXIT\_SUCCESS);

printf("I will not be printed**\n**");

}

**return** 0;

}

**Output**

Value of errno: 2

Error opening the file: No such file or directory

Error printed by perror: No such file or directory

**Explanation:** The code attempts to open a file using fopen(). If the file doesn’t exist, fopen() returns NULL, and the program prints the error using errno, strerror(), and perror() to provide detailed information about the failure. The program then exits with a failure status using exit(EXIT\_FAILURE). If the file is successfully opened, it is closed and the program exits with a success status (exit(EXIT\_SUCCESS)), but no further code is executed after the exit() calls.

**Divide by Zero Errors**

Handling divide by zero errors is essential to avoid program crashes or undefined behavior. You can check for a zero divisor before performing division to prevent this error.

**Example**

#include *<stdio.h>*

int main() {

int num1 = 10, num2 = 0;

**if** (num2 == 0) {

printf("Error: Division by zero is not allowed**\n**");

} **else** {

printf("Result: %d**\n**", num1 / num2);

}

**return** 0;

}

**Output**

Error: Division by zero is not allowed

**Explanation:**The program checks if the divisor num2 is zero before performing the division. If it is, it prints an error message instead of performing the division.