# Error Handling in C

In this tutorial, you will learn certain additional features offered by C to implement error handling though the C language does not provide direct support.

## What is error handling?

Error handling and debugging is an important part of programming. Error handling is the process of detecting and resolving the unwanted stress that interrupts normal operations. Basically, error handling involves two different activities:

1. **Error Detection**: The process of discovering the error that has occurred
2. **Error Recovery**: The process of taking measures to rectify the error

In most of the modern language, the programmers anticipate the possibility of error in advance and they have incorporated some exception codes in the program to tackle the error addressing both detection and recovery in one go. Unfortunately in C routines, this feature is not available and so we end up in need of additional functions that support error handling

## What is errno?

One such additional feature provided by C to handle the error is **error number** in short represented as **errno**. If any function call fails this pre-defined global variable `errno` returns values like '-1' or 'NULL' to notify the error status to the programmer. All of its whereabouts are written inside the header file `error.h`. It is always recommended to initialize the variable as '0' for quick recognition. So during program startup, the value of errno will be zero. When the compilation is over, the programmer can note the returned value and take the necessary steps.

The table below lists the errno and its corresponding type of error.

| 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 | Argument list too long |
| 8 | Exec format error |
| 9 | Bad file number |
| 10 | No child processes |
| 11 | Try again |
| 12 | Out of memory |
| 13 | Permission denied |

## How is perror() and strerror() associated with errno

To display the error message associated with `errno` the following two functions are used in C:

### 1. perror()

This function displays the string passed followed by a colon and space and then the description of the error code stored in the errno.

When we write codes to open or read a file, the compile instructs the Operating System to do so. But sometimes it may fail if the file does not exist or is corrupted. Then the library file `error.h` will save an error code in `errno`. Using `perror`, we can show the desired text message instead of a mere code. Look at the following program:

```
#include <stdlib.h>

int main()
{

  FILE *fp;

  fp = fopen("my_file.txt", "r");
  if (fp == NULL)
  {
     printf("Value of errno: %d\n ", errno);
     perror("Message from perror");
  }
}
```

Here the compiler will look for the file 'my_file.txt'. If not found, the file pointer will return a null value and perror will display the message " No such file or directory". On the screen, we will see the passed string with perror(), then a colon and space followed by this message.

**Output:**

```
Value of errno: 2
Message from perror: No such file or directory
```

### 2.strerror()

This is another function defined inside the string.h header file that returns a pointer to the system generated error message which is associated with the errno. Examine the following example to understand the core difference between these two functions.

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

int main()
{

      FILE *fp;

       fp = fopen("my_file.txt", "r");
        if (fp == NULL)
         {
            printf("Value of errno: %d\n ", errno);
            perror("Message from perror");
            printf("The error message by strerror is : %s\n", strerror(errno));
         }
        else{
            fclose(fp);
         }
}
```

Here both of the functions perror and strerror will print the same message "No such file or directory". The difference is that the former represents the textual format of errno, whereas the later does this by passing the error number as an argument of the function.

```
Value of errno: 2
Message from perror:No such file or directory
The error message by strerror is : No such file or directory
```

## Division by zero error

We encounter this type of error very often, especially involving complex mathematical calculations. When writing codes for calculators or banking software, we may forget that the user may input data for the divisor as zero. As we know that any number divided by zero results infinite, so the program will flag an error if this happens.

So the best practice is to route all the cases where divisors are zero in a separate block of codes. This will flag an appropriate error message and the user will easily understand what went wrong. In the following program, we will try to divide 1 by 0 and flag the error using the strerror function.

```
#include <stdio.h>
#include <stdlib.h>
main()
 {
        int a = 1;
        int b = 0;
        int c;
        if( b == 0)
        {
                fprintf(stderr, " Zero can not be a divisor...\n");
                exit(-1);
         }

    c = a / b;
   fprintf(stderr, "The result of the division is %d\n", c );
    exit(0);
}
```

**Output:**

```
Zero cannot be a divisor…
```

## Program exit status

Generally, there are two possible outcomes of a program, success or failure. It is a good practice among the programmers to return exit with a value '0' when it runs successfully. Otherwise '-1' is returned as a representation of negative results. When the input data is not correct, execution of the rest of the codes is of no use. So it will be better to exit the program with a message.

Here is a division program:

```
#include <stdio.h>
#include <stdlib.h>
main()
  {
     int a, b, c;
     printf(" Enter the dividend");
     scanf("%d",&a);
     printf ("\n Enter the divisor");
     scanf("%d",&b);
     if( b== 0)
     {
          fprintf(stderr, "\nZero cannot be a divisor! ");
          exit(EXIT_FAILURE);
     }
     c=a/b;
     fprintf(stderr, " \nTHE RESULT IS : %d", c );
     exit(EXIT_SUCCESS);
}
```

If zero is input as a divisor, the program will exit with the defined error message " Zero cannot be a divisor!". Otherwise, it will display the right quotient.

**Output:**

```
Enter the dividend  40
Enter the divisor 2
THE RESULT IS : 20
                OR
Enter the dividend 4
Enter the divisor 0
Zero cannot be a divisor!
```