C - ERROR HANDLING

As such C programming does not provide direct support for error handling but being a system programming language, it provides you access at lower level in the form of return values. Most of the C or even Unix function calls return -1 or NULL in case of any error and sets an error code **errno** is set which is global variable and indicates an error occurred during any function call. You can find various error codes defined in <error.h> header file.

So a C programmer can check the returned values and can take appropriate action depending on the return value. As a good practice, developer should set errno to 0 at the time of initialization of the program. A value of 0 indicates that there is no error in the program.

The errno, perror() and strerror()

The C programming language provides **perror()** and **strerror()** functions which can be used to display the text message associated with **errno**.

- The **perror()** function displays the string you pass to it, followed by a colon, a space, and then the textual representation of the current errno value.
- The **strerror()** function, which returns a pointer to the textual representation of the current errno value.

Let's try to simulate an error condition and try to open a file which does not exist. Here I'm using both the functions to show the usage, but you can use one or more ways of printing your errors. Second important point to note is that you should use **stderr** file stream to output all the errors.

```
#include <stdio.h>
#include <errno.h>
#include <string.h>

extern int errno ;

int main ()
{
   FILE * pf;
   int errnum;
   pf = fopen ("unexist.txt", "rb");
   if (pf == NULL)
   {
      errnum = errno;
      fprintf(stderr, "Value of errno: %d\n", errno);
      perror("Error printed by perror");
      fprintf(stderr, "Error opening file: %s\n", strerror( errnum ));
   }
   else
   {
      fclose (pf);
   }
   return 0;
}
```

When the above code is compiled and executed, it produces the following result:

```
Value of errno: 2
Error printed by perror: No such file or directory
Error opening file: No such file or directory
```

Divide by zero errors

It is a common problem that at the time of dividing any number, programmers do not check if a divisor is zero and finally it creates a runtime error.

The code below fixes this by checking if the divisor is zero before dividing:

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

main()
{
   int dividend = 20;
   int divisor = 0;
   int quotient;

   if( divisor == 0){
      fprintf(stderr, "Division by zero! Exiting...\n");
      exit(-1);
   }
   quotient = dividend / divisor;
   fprintf(stderr, "Value of quotient : %d\n", quotient );

   exit(0);
}
```

When the above code is compiled and executed, it produces the following result:

```
Division by zero! Exiting...
```

Program Exit Status

It is a common practice to exit with a value of EXIT_SUCCESS in case of programming is coming out after a successful operation. Here, EXIT_SUCCESS is a macro and it is defined as 0.

If you have an error condition in your program and you are coming out then you should exit with a status EXIT_FAILURE which is defined as -1. So let's write above program as follows:

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

main()
{
   int dividend = 20;
   int divisor = 5;
   int quotient;

   if( divisor == 0){
      fprintf(stderr, "Division by zero! Exiting...\n");
      exit(EXIT_FAILURE);
   }
   quotient = dividend / divisor;
   fprintf(stderr, "Value of quotient : %d\n", quotient );

   exit(EXIT_SUCCESS);
}
```

When the above code is compiled and executed, it produces the following result:

```
Value of quotient : 4
```