File Management & Input -Output

{C}
Programming

File management is what you have, and how you want to manipulate it. - Anonymous

Formatted I/O in C

- The ***I/O procedure*** is known as ***"formatted I/O"***. It enables you to read or write data in a certain format. ***Printf()*** and ***scanf()*** are two examples of C routines that handle ***formatted I/O***. The type and format of the data to be read or written are specified by ***format strings***, which are used by these operations. The program's execution replaces the placeholders for the data found in the format strings with the actual data.

Example:

Let's examine an illustration of formatted I/O in C:

```
#include <stdio.h>

int main() {
   char name[20];
   int age;

printf("Enter your name: ");
scanf("%s", name);

printf("Enter your age: ");
scanf("%d", &age);

printf("Your name is %s and your age is %d\n", name, age);

   return 0;
}
```

Output:

```
Enter your name: Avi
Enter your age: 22
Your name is Avi and your age is 22
```

Why File Management?

- In real life, we want to store data permanently so that later we can retrieve it and reuse it.
- A file is a collection of characters stored on a secondary storage device like hard disk, or pen drive.
- There are two kinds of files that programmer deals with:
  - Text Files are human readable and it is a stream of plain English characters
  - Binary Files are computer readable, and it is a stream of processed characters and ASCII symbols

Text File

Hello, this is a text file. Whatever written here can be read easily without the help of a computer.

Binary File

```
11010011010100010110111010
10111010111010011010100010
11011101010111010111010011
```

File Opening Modes

- We can perform different operations on a file based on the file opening modes

Mode	Description
r	Open the file for reading only. If it exists, then the file is opened with the current contents; otherwise an error occurs.
w	Open the file for writing only. A file with specified name is created if the file does not exists. The contents are deleted, if the file already exists.
a	Open the file for appending (or adding data at the end of file) data to it. The file is opened with the current contents safe. A file with the specified name is created if the file does not exists.
r+	The existing file is opened to the beginning for both reading and writing.
w+	Same as w except both for reading and writing.
a+	Same as a except both for reading and writing.

Note: The main difference is w+ truncate the file to zero length if it exists or create a new file if it doesn't. While r+ neither deletes the content nor create a new file if it doesn't exist.

File Handling Functions

- Basic file operation performed on a file are opening, reading, writing, and closing a file.

Syntax	Description
`fp=fopen(file_name, mode);`	This statement opens the file and assigns an identifier to the FILE type pointer `fp`. Example: `fp = fopen("printfile.c","r");`
`fclose(filepointer);`	Closes a file and release the pointer. Example: `fclose(fp);`
`fprintf(fp, “control string”, list);`	Here `fp` is a file pointer associated with a file. The control string contains items to be printed. The list may includes variables, constants and strings. Example: `fprintf(fp, "%s %d %c", name, age, gender);`

File Handling Functions

Syntax	Description
`fscanf(fp, “control string”, list);`	Here `fp` is a file pointer associated with a file. The control string contains items to be printed. The list may includes variables, constants and strings. Example: `fscanf(fp, "%s %d", &item, &qty);`
`int getc( FILE *fp);`	`getc()` returns the next character from a file referred by `fp`; it require the `FILE` pointer to tell from which file. It returns `EOF` for end of file or error. Example: `c = getc(fp);`
`int putc(int c, FILE *fp);`	`putc()` writes or appends the character `c` to the `FILE` `fp`. If a `putc` function is successful, it returns the character written, `EOF` if an error occurs. Example: `putc(c, fp);`

File Handling Functions

Syntax	Description
`int getw(FILE *pvar);`	`getw()` reads an integer value from `FILE` pointer `fp` and returns an `integer`. Example: `i = getw(fp);`
`putw(int, FILE *fp);`	putw writes an integer value read from terminal and are written to the `FILE` using `fp`. Example: `putw(i, fp);`
`EOF`	`EOF` stands for “End of File”. `EOF` is an integer defined in `<stdio.h>` Example: `while(ch != EOF)`

Write a C program to display content of a given file.

Program

```
 1  #include <stdio.h>
 2  void main()
 3  {
 4      FILE *fp; //p is a FILE type pointer
 5      char ch; //ch is used to store single character
 6      fp = fopen("file1.c","r"); //open file in read mode and store file pointer in p
 7      do {  //repeat step 9 and 10 until EOF is reached
 8          ch = getc(fp); //get character pointed by p into ch
 9          putchar(ch); //print ch value on monitor
10      }while(ch != EOF); //condition to check EOF is reached or not
11      fclose(fp); //free up the file pointer pointed by fp
12  }
13
```

Write a C program to copy a given file.

Program

```c
#include <stdio.h>
void main()
{
    FILE *fp1, *fp2; //p and q is a FILE type pointer
    char ch; //ch is used to store temporary data
    fp1 = fopen("file1.c","r"); //open file “file1.c” in read mode
    fp2 = fopen("file2.c","w"); //open file “file2.c” in write mode
    do { //repeat step 9 and 10 until EOF is reached
        ch = getc(fp1); //get character pointed by p into ch
        putc(ch, fp2); //print ch value into file, pointed by pointer q
    }while(ch != EOF); //condition to check EOF is reached or not
    fclose(fp1); //free up the file pointer p
    fclose(fp2); //free up the file pointer q
    printf("File copied successfully...");
}
```

File Positioning Functions

- `fseek`, `ftell`, and `rewind` functions will set the file pointer to new location.
- A subsequent read or write will access data from the new position.

Syntax	Description
`fseek(FILE *fp, long offset, int position);`	`fseek()` function is used to move the file position to a desired location within the file. **`fp`** is a `FILE` pointer, **`offset`** is a value of datatype `long`, and **`position`** is an `integer` number. **Example**: /* Go to the end of the file, past the last character of the file */ `fseek(fp, 0L, 2);`
`long ftell(FILE *fp);`	`ftell` takes a file pointer and returns a number of datatype `long`, that corresponds to the current position. This function is useful in saving the current position of a file. **Example**: /* n would give the relative offset of the current position. */ `n = ftell(fp);`

File Positioning Functions

Syntax	Description
`rewind(fp);`	`rewind()` takes a file pointer and resets the position to the start of the file. **Example**: /* The statement would assign 0 to n because the file position has been set to the start of the file by rewind. */ `rewind(fp);`

Write a C program to count lines, words, tabs, and characters

Program

```c
#include <stdio.h>
void main()
{
    FILE *p;
    char ch;
    int ln=0,t=0,w=0,c=0;
    p = fopen("text1.txt","r");
    ch = getc(p);
    while (ch != EOF) {
        if (ch == '\n')
            ln++;
        else if(ch == '\t')
            t++;
        else if(ch == ' ')
            w++;
        else
```

Program (contd.)

```c
                c++;

            ch = getc(p);
    }
    fclose(p);
    printf("Lines = %d, tabs = %d, words = %d, characters = %d\n",ln, t, w, c);
}
```

Output

```
Lines = 22, tabs = 0, words = 152, characters = 283
```

C - Error Handling

- 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 set an error code **errno**. It is set as a global variable and indicates an error occurred during any function call. You can find various error codes defined in <error.h> header file.
- 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**.

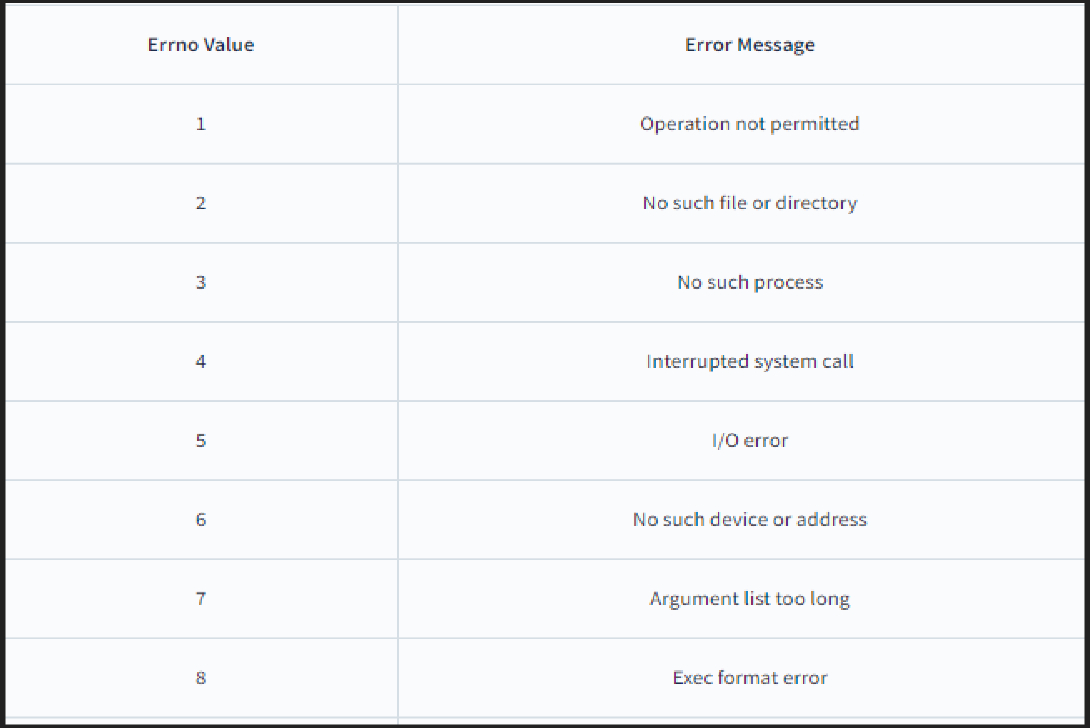
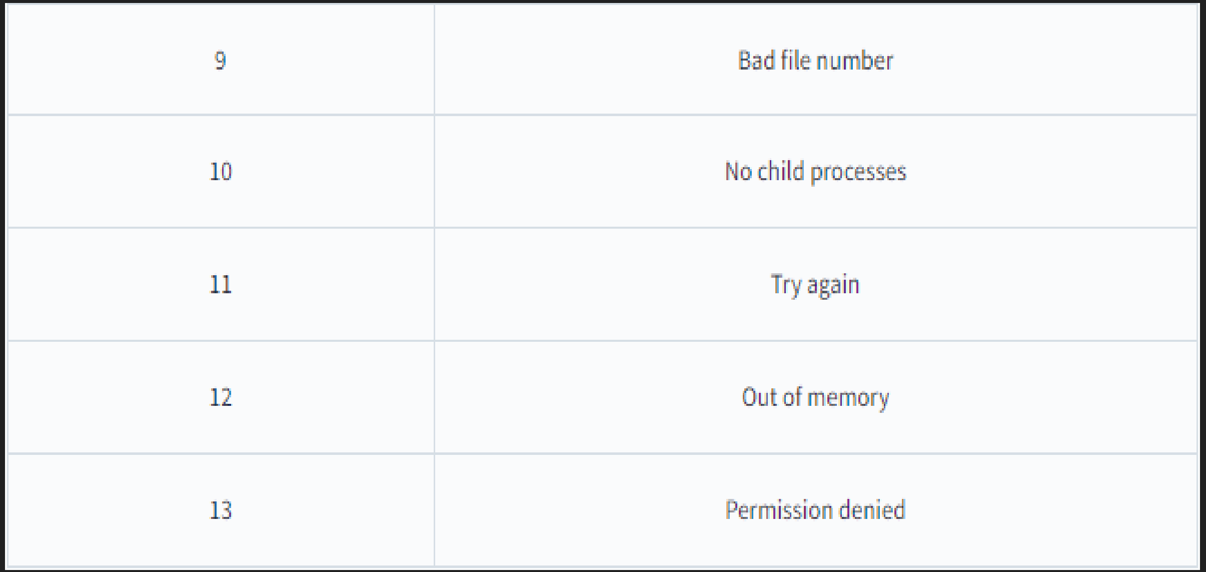
The **perror()** function displays the string you pass to it, followed by a colon, a space, and then the textual representation of the current errno value.

The **strerror()** function, which returns a pointer to the textual representation of the current errno value.

- Second important point to note is that you should use **stderr** file stream to output all the errors.

What is Errno?

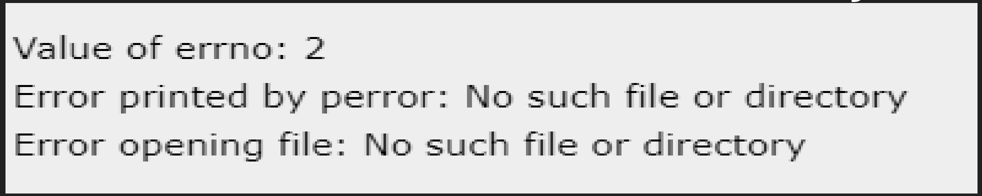
- When we call a function in C language, a variable is automatically initialized with a numeric value and we can use that to identify the type of error if encountered while writing the code. This variable is called **errno value.** It is a global variable that is defined in **errno.h** header file. There are a total of 13 errno values in C language and each errno has an error message associated with it. These are illustrated below: **(Different types of possible error messages):**https://www.scaler.com/topics/c/error-handling-in-c/

Errno Value	Error Message
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

Example(Errno)

```
#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;
}
```



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

Continue...

2. strerror()

strerror() function is contained in **string.h** header file.

It takes **'errno'** as an argument and returns the string error message of currently passed errno.

Syntax:

```
char *strerror(int errnum)
```

3. ferror()

ferror() function is contained in **stdio.h** header file.

This function basically checks for error in the file stream. It returns zero value if there is no error or else, it returns a positive non-zero value in case of error.

File pointer stream is passed as an argument to the function. It will check for the error until the file is closed or we call clearerr() function.

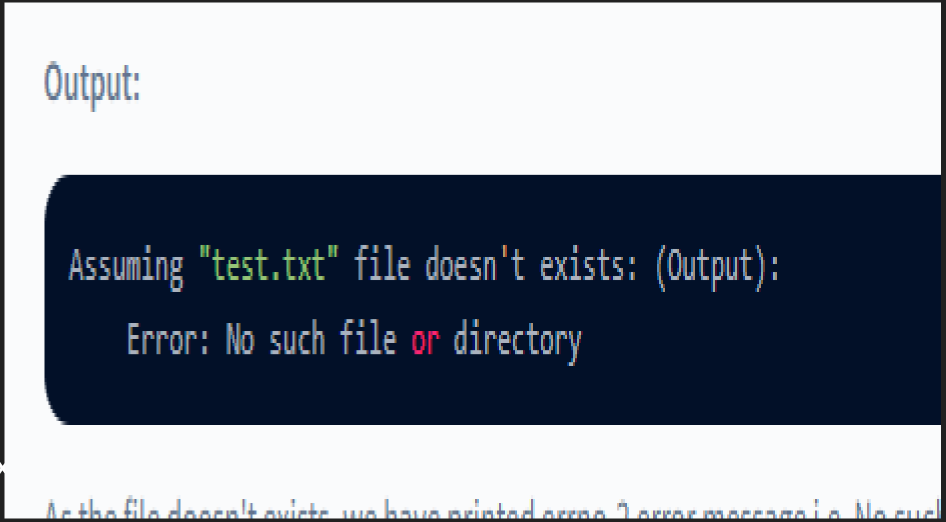
To identify the type of error, we can further use perror() function.

Syntax:

```
int ferror(FILE *stream);
```

Example(strerror)

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

int main(){
    FILE* fp;
    fp = fopen("test.txt","r");

    if(fp==NULL){//Error handling in case if the file doesn't ex
        printf("Error: %s\n",strerror(errno));
        //errno passed as an argument to display respective order error message
    }
    fclose(fp);
    return 0;
}
```



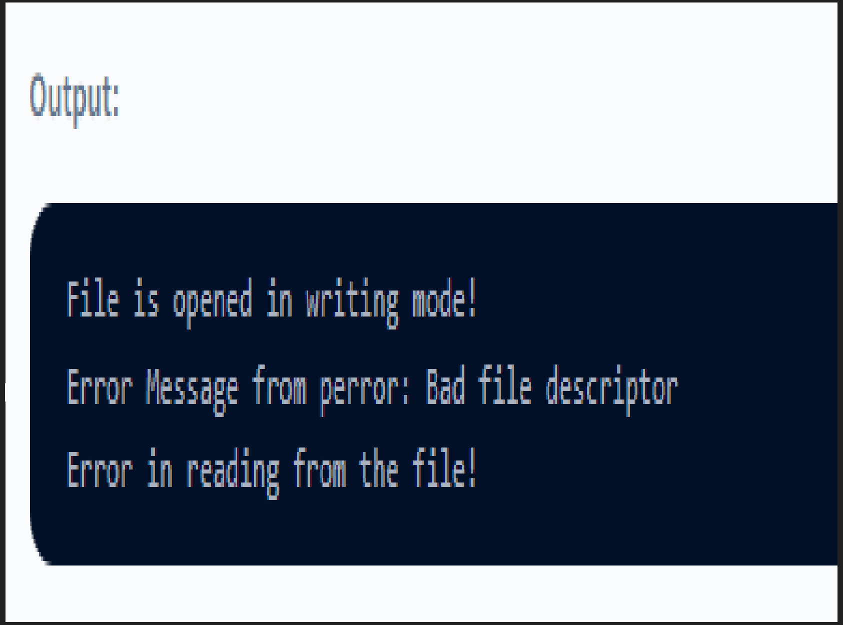
Output:

```
Assuming "test.txt" file doesn't exists: (Output):
    Error: No such file or directory
```

As the file doesn't exists, we have printed errno 2 error message i.e. No such

Example()

```
#include <stdio.h>

int main(){
   FILE *fp;
   fp = fopen("test.txt","w");

   char ch = fgetc(fp);  //Trying to read data, despite of writing mode ope
   if(ferror(fp)){      //Error detected in the file stream pointer
      printf("File is opened in writing mode!");
      printf("\nError in reading from the file!");
      perror("Error Message from perror");
      //Identifying the type of error using perror() function
   }
   fclose(fp);
   return(0);
}
```

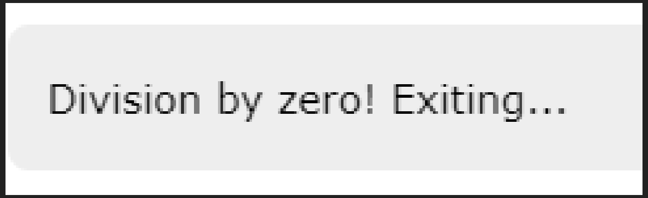

Output:

```
File is opened in writing mode!
Error Message from perror: Bad file descriptor
Error in reading from the file!
```

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.

```
#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);
}
```



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

Continue...

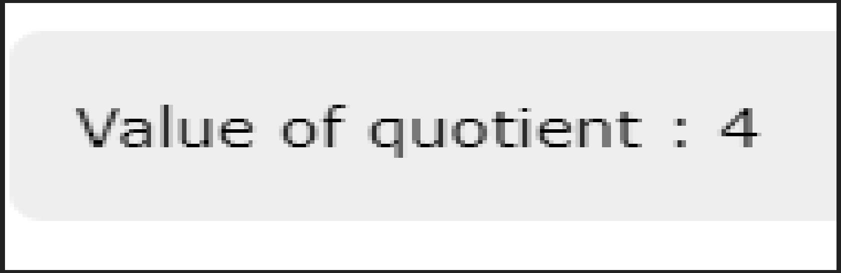
- **Program Exit Status**

It is a common practice to exit with a value of EXIT_SUCCESS in case of program 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 −

Example

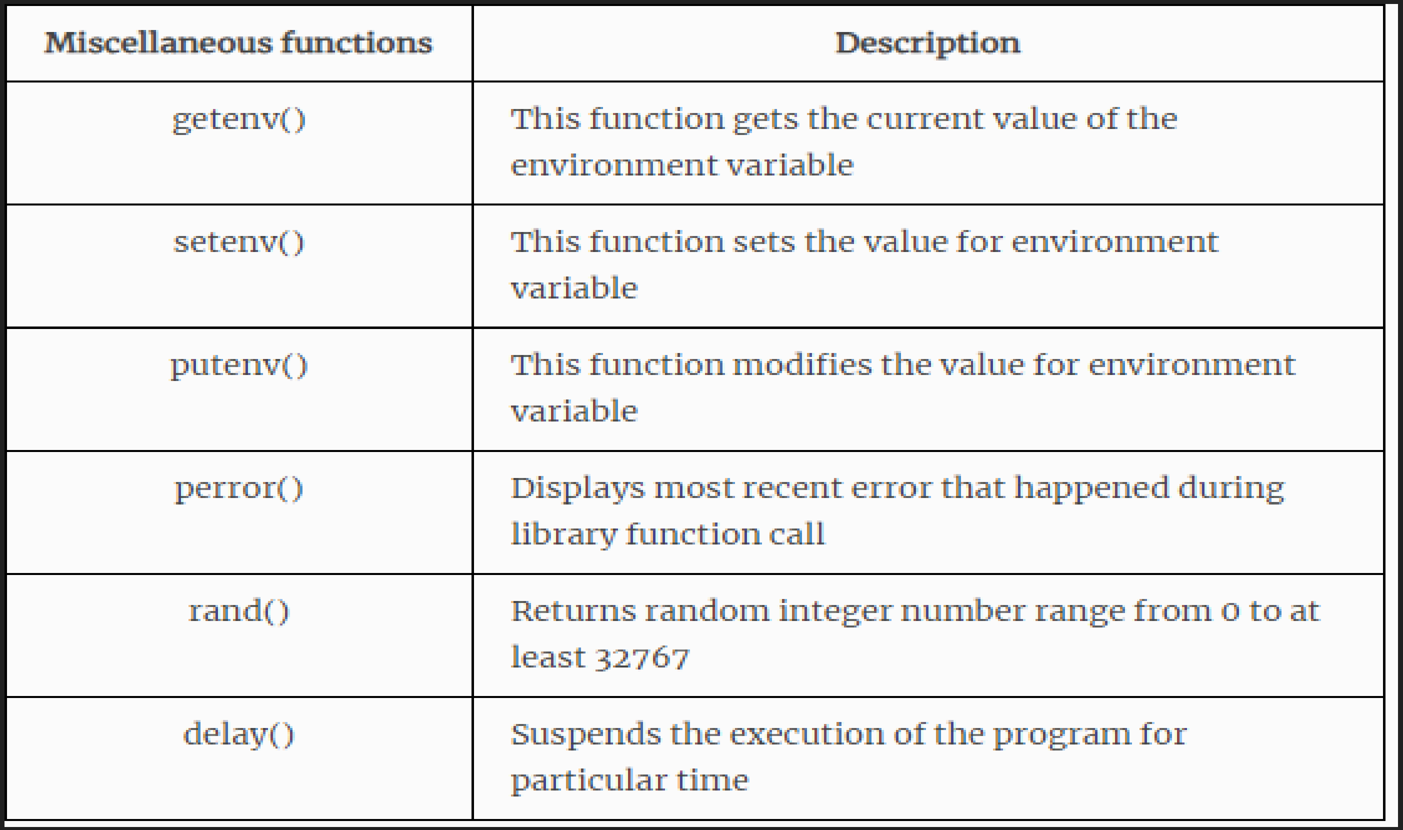
```
#include <stdio.h>
#include <stdlib.h>
main() {
   int dividend = 20;
   int divisor = 5;
   int quotient;
   if( divisor == 0) {
      fprintf(stderr, "Division by zero! Exiting...\n");
      exit(EXIT_FAILURE);
   }
   quotient = dividend / divisor;
   fprintf(stderr, "Value of quotient : %d\n", quotient );
   exit(EXIT_SUCCESS);
}
```



```
Value of quotient : 4
```

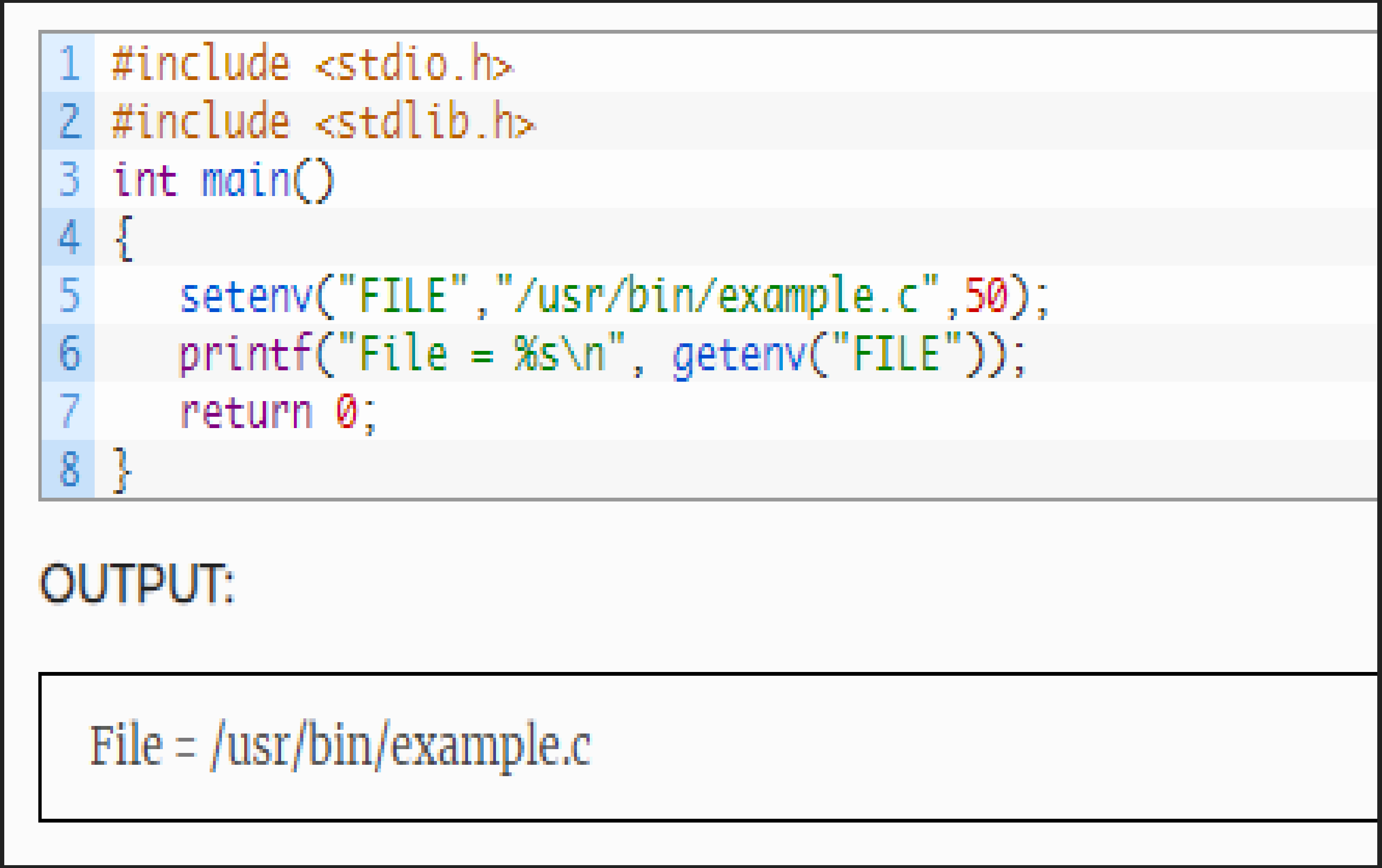
C – Miscellaneous functions

- Miscellaneous functions perform a variety of operations and return specific information or values. The miscellaneous functions are summarized in the table below.



Miscellaneous functions	Description
getenv()	This function gets the current value of the environment variable
setenv()	This function sets the value for environment variable
putenv()	This function modifies the value for environment variable
perror()	Displays most recent error that happened during library function call
rand()	Returns random integer number range from 0 to at least 32767
delay()	Suspends the execution of the program for particular time

EXAMPLE PROGRAM FOR GETENV() &SETENV() FUNCTION IN C:

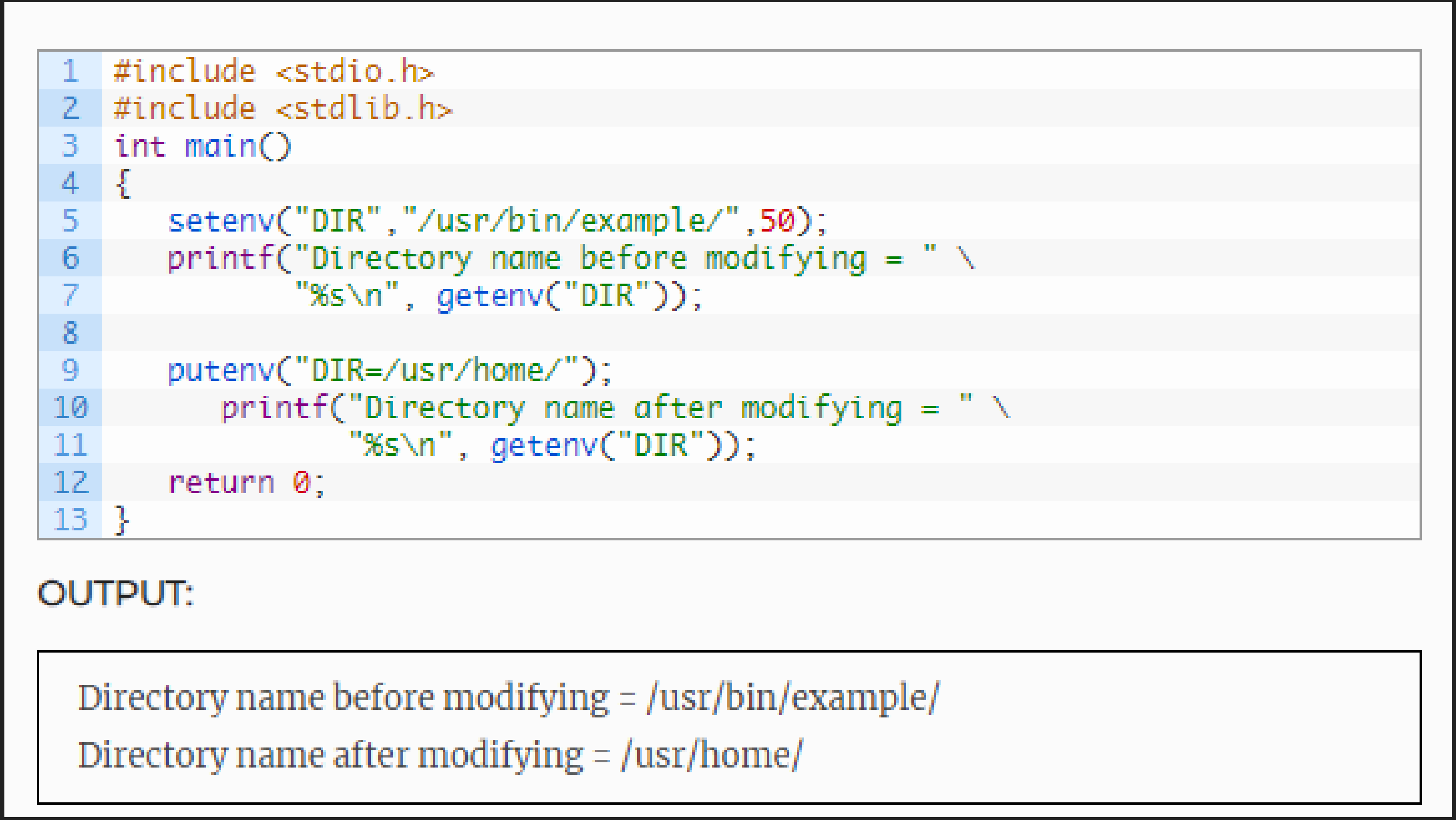


```
#include <stdio.h>
#include <stdlib.h>
int main()
{
   setenv("FILE","/usr/bin/example.c",50);
   printf("File = %s\n", getenv("FILE"));
   return 0;
}
```

OUTPUT:

```
File = /usr/bin/example.c
```

EXAMPLE PROGRAM FOR PUTENV() FUNCTION IN C:

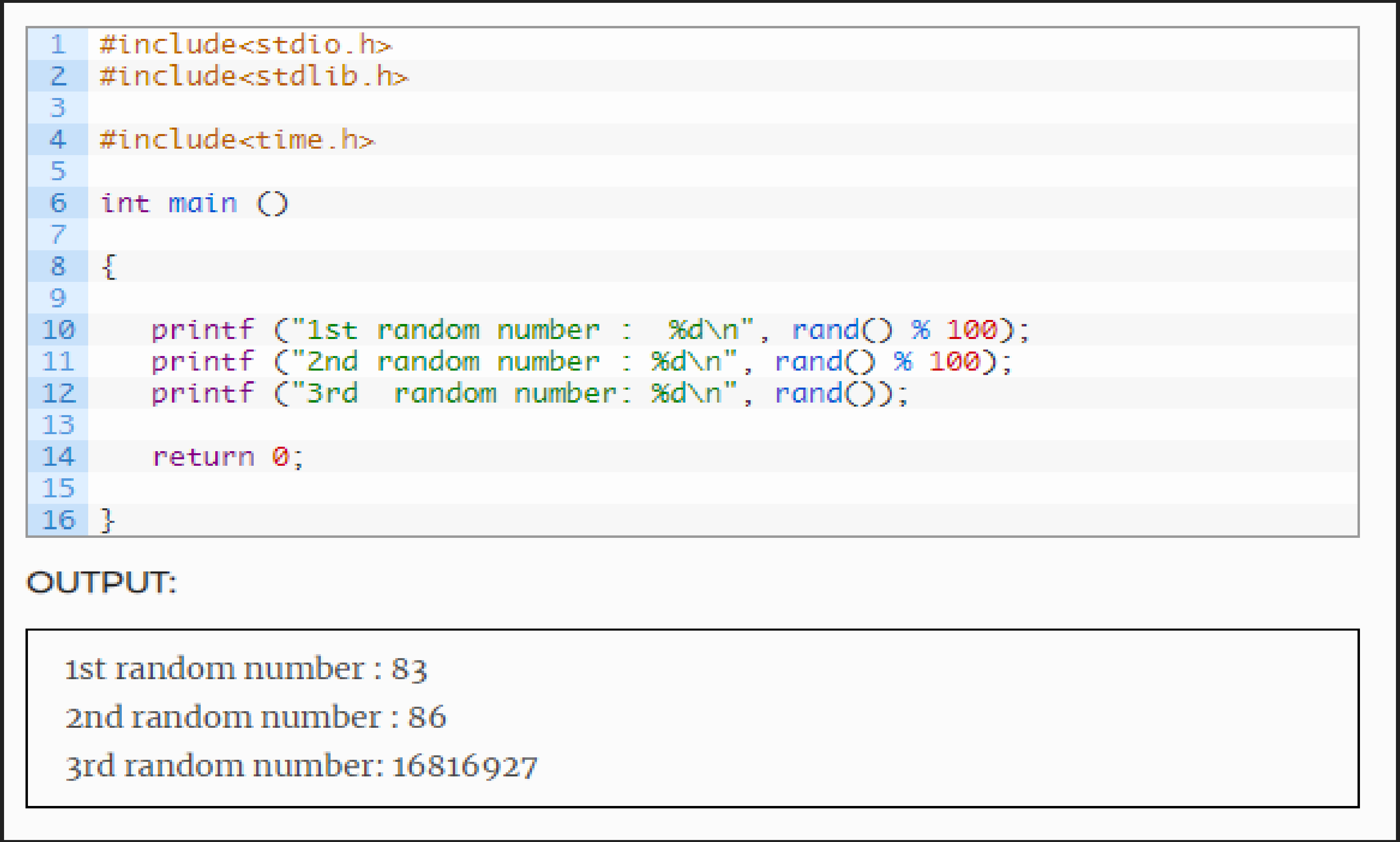


```c
#include <stdio.h>
#include <stdlib.h>
int main()
{
   setenv("DIR","/usr/bin/example/",50);
   printf("Directory name before modifying = " \
          "%s\n", getenv("DIR"));

   putenv("DIR=/usr/home/");
      printf("Directory name after modifying = " \
             "%s\n", getenv("DIR"));
   return 0;
}
```

OUTPUT:

```
Directory name before modifying = /usr/bin/example/
Directory name after modifying = /usr/home/
```

EXAMPLE PROGRAM FOR RAND() FUNCTION IN C:

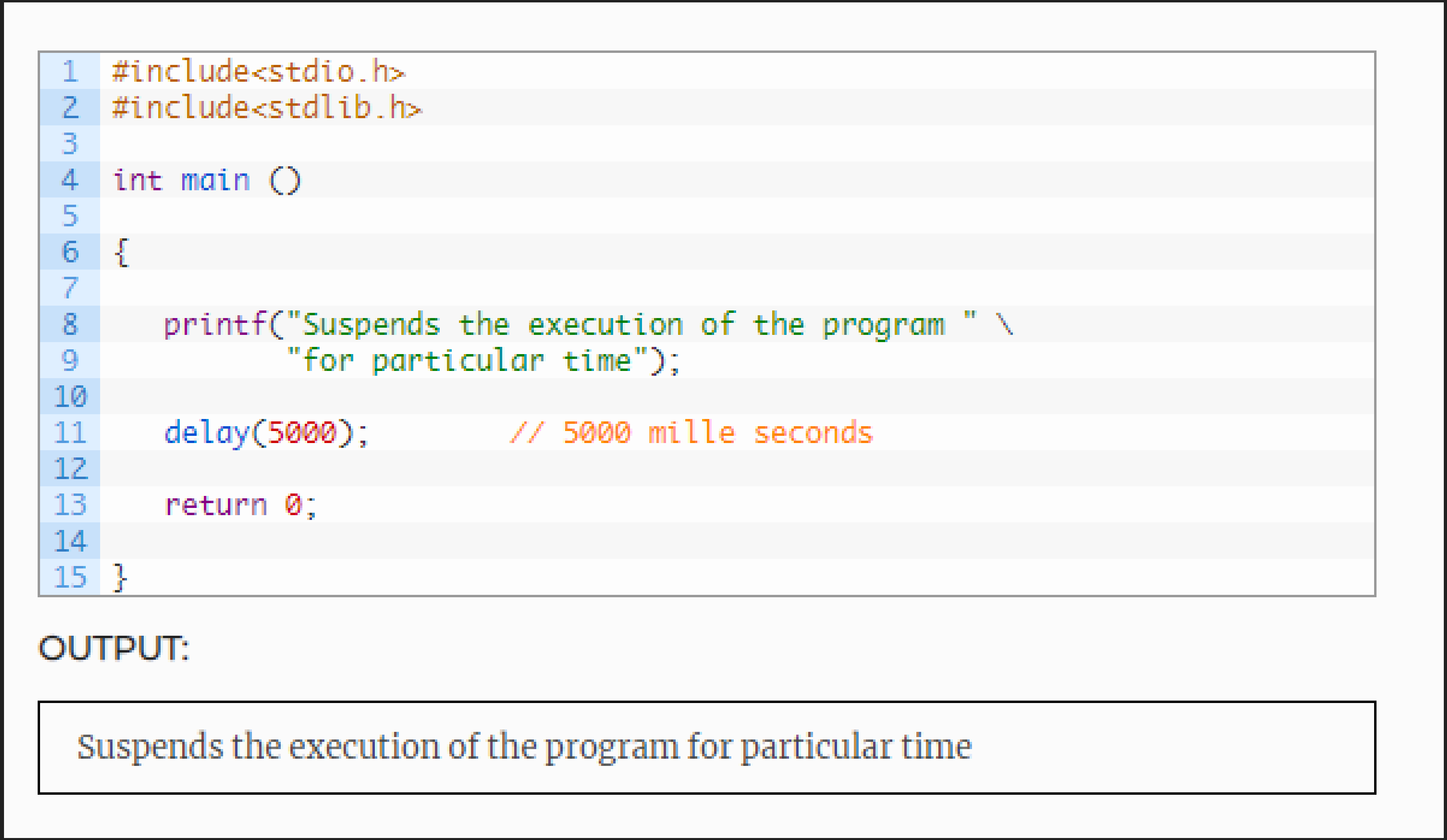


```c
#include<stdio.h>
#include<stdlib.h>

#include<time.h>

int main ()

{

   printf ("1st random number :  %d\n", rand() % 100);
   printf ("2nd random number : %d\n", rand() % 100);
   printf ("3rd  random number: %d\n", rand());

   return 0;

}
```

OUTPUT:

```
1st random number : 83
2nd random number : 86
3rd random number: 16816927
```

EXAMPLE PROGRAM FOR DELAY() FUNCTION IN C:



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

int main ()

{

   printf("Suspends the execution of the program " \
           "for particular time");

   delay(5000);         // 5000 mille seconds

   return 0;

}
```

OUTPUT:

```
Suspends the execution of the program for particular time
```

Practice Programs

1) Write a C program to write a string in file.
2) A file named data contains series of integer numbers. Write a C program to read all numbers from file and then write all the odd numbers into file named "odd" and write all even numbers into file named "even". Display all the contents of these file on screen.
3) Write a C program to read name and marks of n number of students and store them in a file.
4) Write a C program to print contents in reverse order of a file.
5) Write a C program to compare contents of two files.
6) Write a C program to copy number of bytes from a specific offset to another file.
7) Write a C program to convert all characters in UPPER CASE of a File.

Thank you