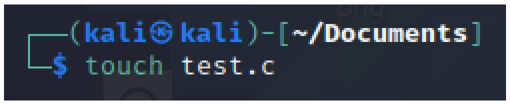
AIR UNIVERSITY

MUHAMMAD ABDULLAH | 221546 | BSCYS-F22-A

STEP 01:

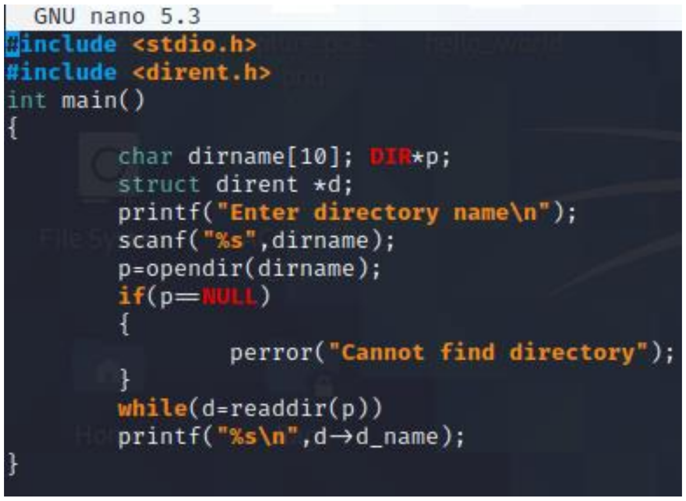
Create a file by using the command **touch** and extension **.c**



```
┌──(kali㉿kali)-[~/Documents]
└─$ touch test.c
```

STEP 02:

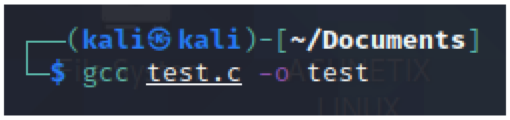
Now write in the following code.



```
  GNU nano 5.3
#include <stdio.h>
#include <dirent.h>
int main()
{
        char dirname[10]; DIR*p;
        struct dirent *d;
        printf("Enter directory name\n");
        scanf("%s",dirname);
        p=opendir(dirname);
        if(p==NULL)
        {
                perror("Cannot find directory");
        }
        while(d=readdir(p))
        printf("%s\n",d->d_name);
}
```

STEP 03:

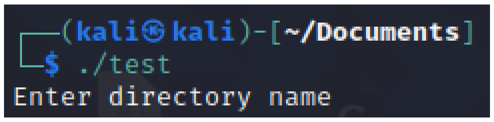
Now make the executable of the file that you have created using the **gcc** command.



```
┌──(kali㉿kali)-[~/Documents]
└─$ gcc test.c -o test
```

STEP 04:

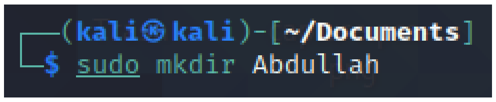
Now run the file using **./<file-name>**



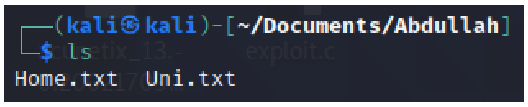
```
┌──(kali㉿kali)-[~/Documents]
└─$ ./test
Enter directory name
```

STEP 05:

Now in order to test this code make a directory and make some empty files in it.



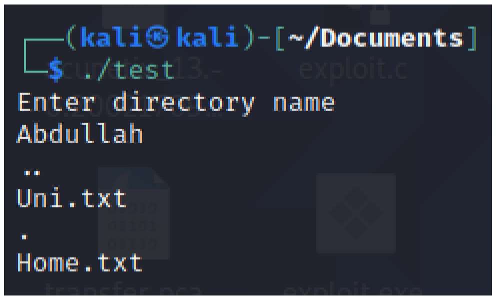
```
┌──(kali㉿kali)-[~/Documents]
└─$ sudo mkdir Abdullah
```



```
┌──(kali㉿kali)-[~/Documents/Abdullah]
└─$ ls
Home.txt  Uni.txt
```

STEP 06:

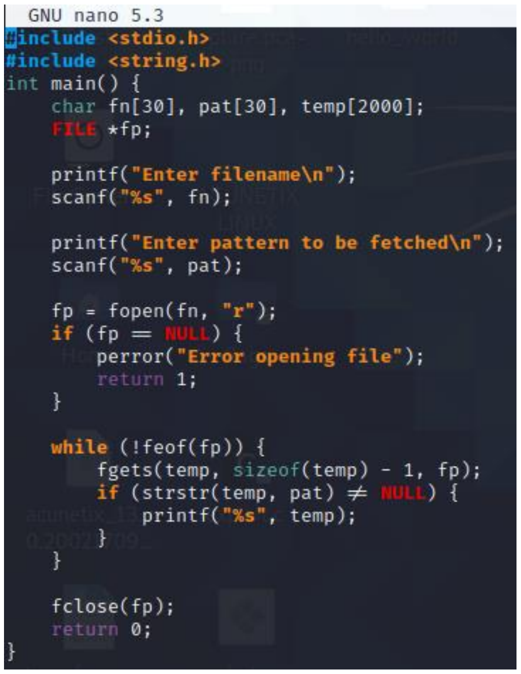
Now run the code and it'll give the following outputs.



```
┌──(kali㉿kali)-[~/Documents]
└─$ ./test
Enter directory name
Abdullah
..
Uni.txt
.
Home.txt
```

STEP 07:

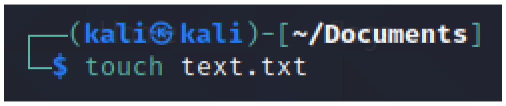
Now write the following code.



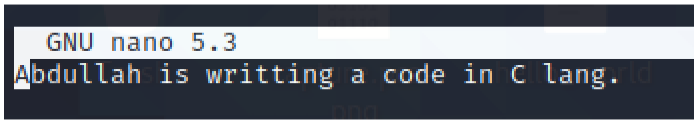
```
  GNU nano 5.3
#include <stdio.h>
#include <string.h>
int main() {
    char fn[30], pat[30], temp[2000];
    FILE *fp;

    printf("Enter filename\n");
    scanf("%s", fn);

    printf("Enter pattern to be fetched\n");
    scanf("%s", pat);

    fp = fopen(fn, "r");
    if (fp == NULL) {
        perror("Error opening file");
        return 1;
    }

    while (!feof(fp)) {
        fgets(temp, sizeof(temp) - 1, fp);
        if (strstr(temp, pat) ≠ NULL) {
            printf("%s", temp);
        }
    }

    fclose(fp);
    return 0;
}
```

STEP 08:

Now in order to run the code correctly create a **.txt** file and then write something in it.



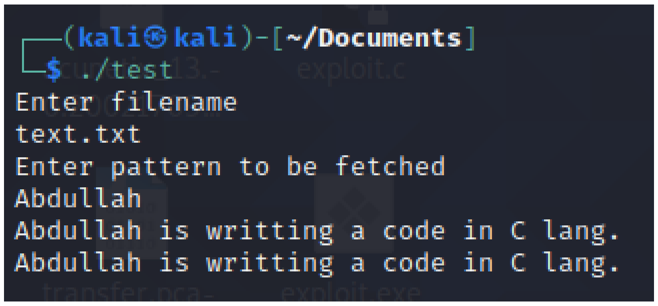
```
┌──(kali㉿kali)-[~/Documents]
└─$ touch text.txt
```



```
  GNU nano 5.3
Abdullah is writting a code in C lang.
```

STEP 09:

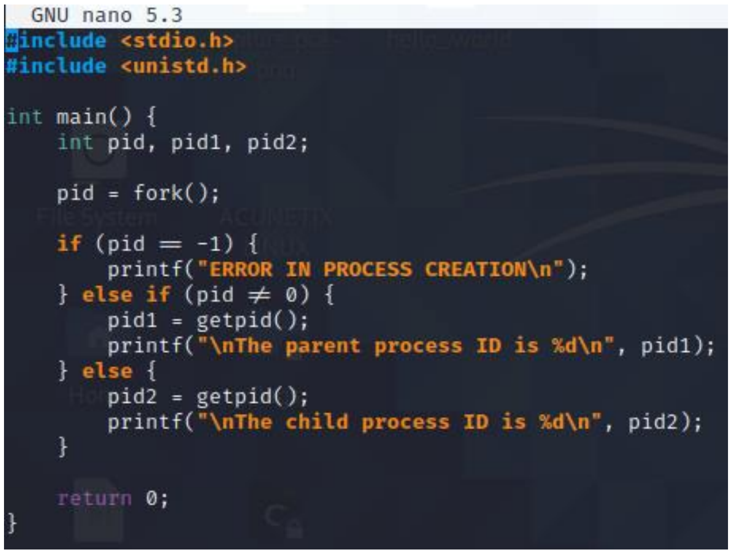
Now run the code and you'll get the respective output as following.



```
┌──(kali㉿kali)-[~/Documents]
└─$ ./test
Enter filename
text.txt
Enter pattern to be fetched
Abdullah
Abdullah is writting a code in C lang.
Abdullah is writting a code in C lang.
```

STEP 10:

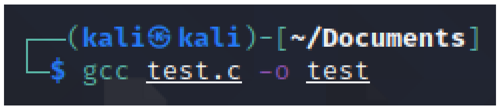
Now write the following code.



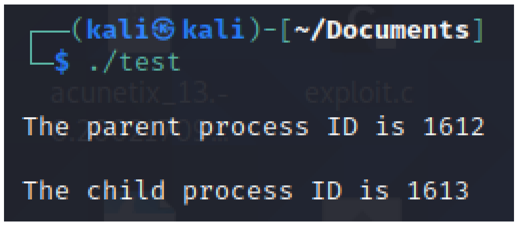
```
  GNU nano 5.3
#include <stdio.h>
#include <unistd.h>

int main() {
    int pid, pid1, pid2;

    pid = fork();

    if (pid == -1) {
        printf("ERROR IN PROCESS CREATION\n");
    } else if (pid != 0) {
        pid1 = getpid();
        printf("\nThe parent process ID is %d\n", pid1);
    } else {
        pid2 = getpid();
        printf("\nThe child process ID is %d\n", pid2);
    }

    return 0;
}
```

STEP 11:

First create the executable and then run the code and then in will assign IDs to the child and the main or parent process.



```
┌──(kali㉿kali)-[~/Documents]
└─$ gcc test.c -o test
```

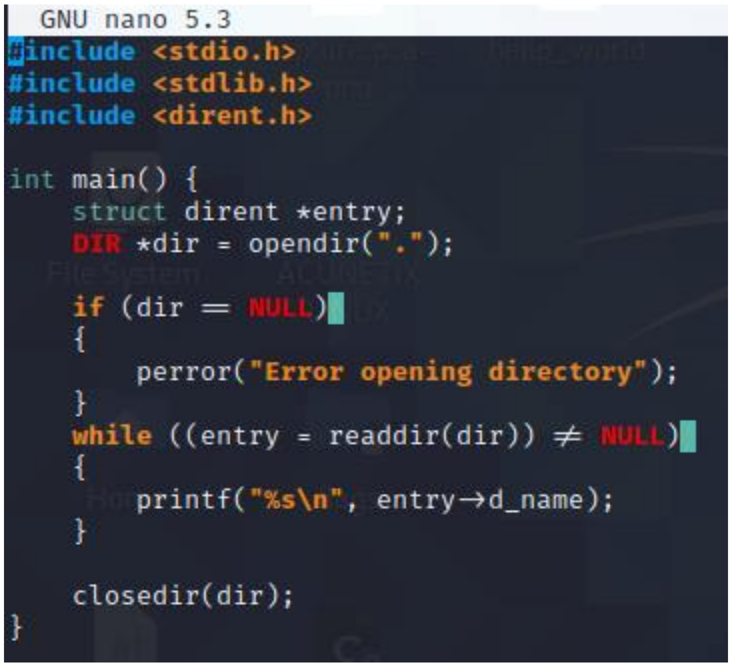


```
┌──(kali㉿kali)-[~/Documents]
└─$ ./test

The parent process ID is 1612

The child process ID is 1613
```

LAB TASK

TASK 01:

Code to replicate **ls** command.



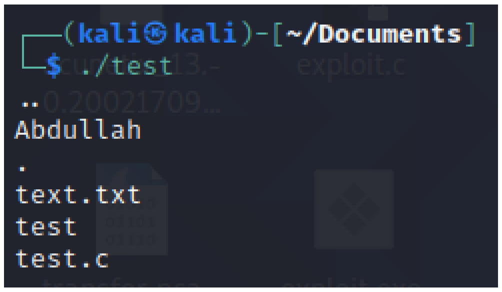
```
  GNU nano 5.3
#include <stdio.h>
#include <stdlib.h>
#include <dirent.h>

int main() {
    struct dirent *entry;
    DIR *dir = opendir(".");
    if (dir == NULL)
    {
        perror("Error opening directory");
    }
    while ((entry = readdir(dir)) != NULL)
    {
        printf("%s\n", entry->d_name);
    }

    closedir(dir);
}
```

CODE EXPLAINATION:

- **stdio.h**: Standard input-output functions.
- **stdlib.h**: Standard library functions like malloc, exit, etc.
- **dirent.h**: Directory entry structure and functions for directory operations.
- Open the current directory **(.)** using **opendir()** function. This function returns a pointer to the directory stream. If the directory cannot be opened, it returns NULL.
- When the directory is open in reads all the contents until the end of directory and print them on the terminal.

OUTPUT:

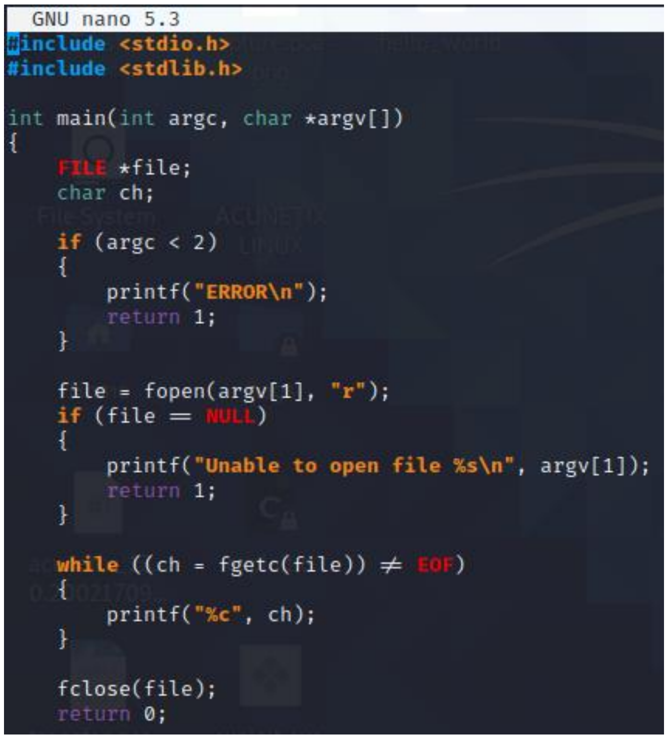
First create the executable and then run the code.



```
┌──(kali㉿kali)-[~/Documents]
└─$ ./test
..
Abdullah
.
text.txt
test
test.c
```

TASK 02:

Code to replicate the **cat** command.



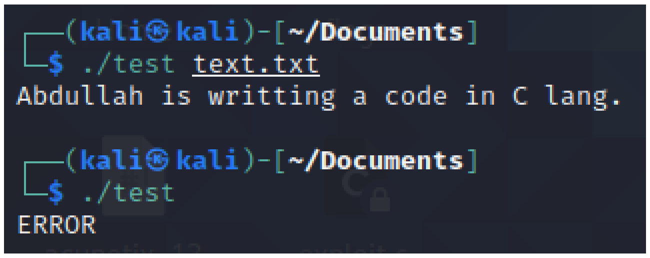
```
  GNU nano 5.3
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    FILE *file;
    char ch;

    if (argc < 2)
    {
        printf("ERROR\n");
        return 1;
    }

    file = fopen(argv[1], "r");
    if (file == NULL)
    {
        printf("Unable to open file %s\n", argv[1]);
        return 1;
    }

    while ((ch = fgetc(file)) != EOF)
    {
        printf("%c", ch);
    }

    fclose(file);
    return 0;
```

CODE EXPLAINATION:

- In this the **main** function of the program takes two parameters: **argc (argument count)** and **argv (argument vector),** which are used to handle command-line arguments.
- In the next part it checks if the user has provided the filename as a command-line argument. If not, it displays error message.
- Next it tries to open the file specified by the user. It uses the **fopen** function, which takes the filename and mode as arguments. In this case, **"r"** mode indicates that the file will be opened for reading.
- Next the loop reads characters from the file one by one using **fgetc** function and prints them to the console until the end of the file **(EOF)** is reached.

OUTPUT:

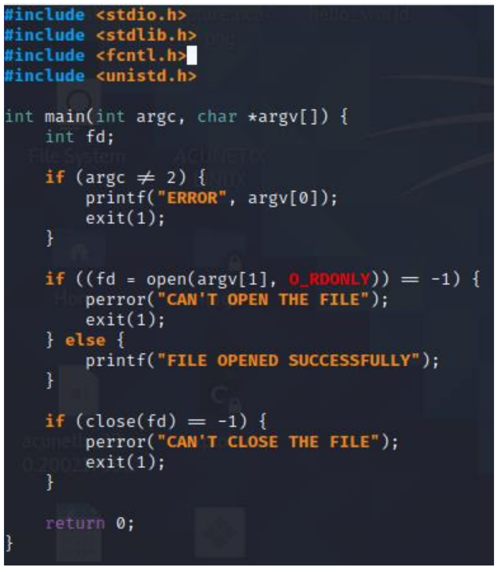
Create the code executable by using the **gcc** command and then run the executable file using **./**



```
┌──(kali㉿kali)-[~/Documents]
└─$ ./test text.txt
Abdullah is writting a code in C lang.

┌──(kali㉿kali)-[~/Documents]
└─$ ./test
ERROR
```

TASK 03:

Write the following code.



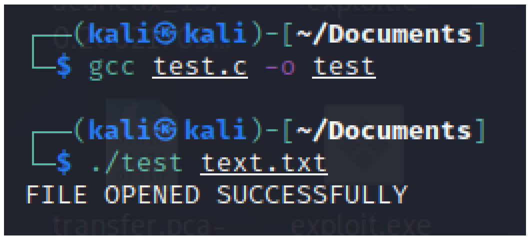
```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>

int main(int argc, char *argv[]) {
    int fd;

    if (argc ≠ 2) {
        printf("ERROR", argv[0]);
        exit(1);
    }

    if ((fd = open(argv[1], O_RDONLY)) = -1) {
        perror("CAN'T OPEN THE FILE");
        exit(1);
    } else {
        printf("FILE OPENED SUCCESSFULLY");
    }

    if (close(fd) = -1) {
        perror("CAN'T CLOSE THE FILE");
        exit(1);
    }

    return 0;
}
```

CODE EXPLAINATAION:

The provided code opens a file named **"file"** for reading using the **open()** system call, checks if the file was opened successfully, prints a message if successful, closes the file using the **close()** system call, and returns 0 to indicate successful execution.

OUTPUT:

Create the code executable by using the **gcc** command and then run the executable file using **./**



```
┌──(kali㉿kali)-[~/Documents]
└─$ gcc test.c -o test

┌──(kali㉿kali)-[~/Documents]
└─$ ./test text.txt
FILE OPENED SUCCESSFULLY
```

****