1. Write a C program to display the file content in reverse order using lseek system call.

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

int main()
{
int fd1, fd2, offset;
char c;
fd1 = open("foo.txt", O_RDONLY);
if(fd1<0)
        printf("OPEN ERROR");

fd2 = open("foorev.txt",O_RDWR);
if(fd2<0)
        printf("OPEN ERROR");

offset = lseek(fd1, 0L, SEEK_END);

while(offset>0)
{
read(fd1, &c, 1);
write(fd2,&c,1);
lseek(fd1,-2,SEEK_CUR);
offset--;
}

close(fd1);
close(fd2);
return 0;
}

//create two files foo.txt and foorev.txt

cat foorev.txt      DLROW OLLEH
cat foo.txt         HELLO WORLD
```

2. Write a C program

a. to read first 20 characters from a file

b. seek to 10th byte *from the beginning* and display 20 characters from there

c. seek 10 bytes ahead from the current file offset and display 20 characters

d. display the file size

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

int main()
{
    int file=0, n;
    char buffer[25];

    if((file=open("testfile.txt",O_RDONLY)) < 0)
            return 1;
    if(read(file,buffer,20) != 20)
            return 1;
    //write(STDOUT_FILENO, buffer, 20);
    printf("\n");

    if(lseek(file,10,SEEK_SET) < 0)
            return 1;
    if(read(file,buffer,20) != 20)
            return 1;
    write(STDOUT_FILENO, buffer, 20);
    printf("\n");

  if(lseek(file,10,SEEK_CUR) < 0)
            return 1;
    if(read(file,buffer,20) != 20)
            return 1;
    write(STDOUT_FILENO, buffer, 20);
    printf("\n");
```

```
        if((n = lseek(file,0,SEEK_END)) <0)
                return 1;

        printf("size of file is %d bytes\n",n);
        close(file);

        return 0;
}
```

```
testfile.txt
a1234567890
b1234567890
c1234567890
d1234567890
e1234567890
f1234567890
```

./a.out

```
0
b1234567890
c12345
4567890
e1234567890

size of file is 72 bytes
```

3. Write a C program to display various details of a file using stat structure (At least 5 fields)

```
#include <unistd.h>
#include <stdio.h>
#include <sys/stat.h>
#include <sys/types.h>

int main(int argc, char **argv)
{
   if(argc != 2)
      return 1;

   struct stat fileStat;
   if(stat(argv[1],&fileStat) < 0)
      return 1;

   printf("Information for %s\n",argv[1]);
   printf("---------------------------\n");
   printf("File Size: \t\t %d bytes\n",(int)fileStat.st_size);
                  printf("Number of Links: \t %d
\n",(int)fileStat.st_nlink);
   printf("File inode: \t\t %d \n", (int)fileStat.st_ino);

   printf("File Permissions: \t");
   printf( (S_ISDIR(fileStat.st_mode)) ? "d" : "-");
   printf( (fileStat.st_mode & S_IRUSR) ? "r" : "-");
   printf( (fileStat.st_mode & S_IWUSR) ? "w" : "-");
   printf( (fileStat.st_mode & S_IXUSR) ? "x" : "-");
   printf( (fileStat.st_mode & S_IRGRP) ? "r" : "-");
   printf( (fileStat.st_mode & S_IWGRP) ? "w" : "-");
   printf( (fileStat.st_mode & S_IXGRP) ? "x" : "-");
   printf( (fileStat.st_mode & S_IROTH) ? "r" : "-");
   printf( (fileStat.st_mode & S_IWOTH) ? "w" : "-");
   printf( (fileStat.st_mode & S_IXOTH) ? "x" : "-");
   printf("\n\n");

printf("The file %s a symbolic link\n",
(S_ISLNK(fileStat.st_mode)) ? "is" : "is not");

   return 0;
}
```

./a.out filetype.c

```
Information for filetype.c
---------------------------
File Size:                766 bytes
Number of Links:  1
```

```
        File inode:                     3156286
        File Permissions:  -rw-rw-r--

        The file is not a symbolic link
```

4. Write a C program to implement ls –li command which list the files in a specified directory. Your program should print 5 attributes of files.

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <dirent.h>
#include <time.h>
#include <sys/stat.h>
int main(int argc,char* argv[])
{
struct dirent *dir;
struct stat mystat;
DIR *dp;
dp = opendir(".");

if(dp)
{
   while(dir = readdir(dp))
            {
            stat(dir->d_name,&mystat);
            // inode mode uid guid access_time
            printf("%ld %o %d %d %s %s \n",
 mystat.st_ino, mystat.st_mode, mystat.st_uid, mystat.st_gid,
ctime(&mystat.st_atime),dir->d_name);
            }
   }
}
```

./a.out

```
7346295 100664 1000 1000 Sun Apr 22 15:26:48 2018
 foo.txt
7346034 100664 1000 1000 Sun Apr 22 15:47:03 2018
 testfile.txt
7346290 40775 1000 1000 Sun Apr 22 16:15:51 2018
 .
7346039 100664 1000 1000 Sun Apr 22 16:10:48 2018
 three.c
7346024 100664 1000 1000 Sun Apr 22 15:42:06 2018
 two.c
```

5. Write a C program to remove empty files from the given directory.

```
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <dirent.h>
int main()
{
int fd,n;
DIR *dp;
struct dirent *dir;
dp = opendir("."); //open current directory

        if(dp)
        {
                while(dir = readdir(dp))
                {
                fd                                        =
        open(dir->d_name,O_RDWR,0777);
                n = lseek(fd,0,SEEK_END);
                        if(!n)
                        {
                                unlink(dir->d_name);
                        }
                }
        }
}
//removes if the file is empty
```

6. Write a C program to demonstrate the creation of soft links and the various properties of hard links

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

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

printf("%d",argc);
        if(argc==3)
        {
                printf("\n %s %s \n", argv[1],argv[2]);
                if((link(argv[1],argv[2]))== 0)
                        printf("Hard link Created! \n");
                else
                        printf("Error in hard link Creation \n");
        }

        else if(argc==4)
        {
                printf("\n %s %s \n", argv[1],argv[2]);
                if((symlink(argv[1],argv[2]))== 0)
                        printf("Soft link Created! \n");
                else
                        printf("Error in soft link Creation \n");
        }
        return 0;
        }
```

```
 ./a.out l4.c hlink1
./a.out l4.c symlink dummy

HARDLINK
$ ./a.out prog.c hlink
Hard linking prog.c and hlink
Hard link created

$ ls -li prog.c hlink
3157142 -rw-rw-r-- 2 guest1 guest1 34 Mar  5 09:21 hlink
3157142 -rw-rw-r-- 2 guest1 guest1 34 Mar  5 09:21 prog.c

SOFTLINK
$./a.out prog.c slink dummy
Soft linking prog.c and slink
Soft link created

$ls -li prog.c hlink slink
3157142 -rw-rw-r-- 2 guest1 guest1 34 Mar  5 09:21 hlink
3157142 -rw-rw-r-- 2 guest1 guest1 34 Mar  5 09:21 prog.c
3157335 lrwxrwxrwx 1 guest1 guest1  6 Mar  5 09:23 slink -> prog.c
```

7. Write a C program to Copy access and modification time of a file to another file using utime function.

```
#include <stdio.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
#include <utime.h>
#include <time.h>
#include <fcntl.h>

int main(int argc,char* argv[]) //copying ctime and mtime of argv[2] to
argv[1]
{
int fd;
struct stat buf1;
struct stat buf2;
struct utimbuf times;

if(stat(argv[1],&buf1)<0)
        printf("Error!\n");
if(stat(argv[2],&buf2)<0)
            printf("Error!\n");

printf("Before Copying ...\n");
printf("Access Time %s\nModification Time
%s\n",ctime(&buf1.st_atime),ctime(&buf1.st_mtime));

times.modtime = buf2.st_mtime;
times.actime = buf2.st_mtime;

            if(utime(argv[1],&times)<0)
            printf("Error copying time \n");

            if(stat(argv[1],&buf1)<0)
            printf("Error!\n");

printf("After Copying ...\n");
printf("Access Time %s\nModification Time
%s\n",ctime(&buf1.st_atime),ctime(&buf1.st_mtime));
}
```

```
$ ls -li three.c six.c
7346518 -rw-rw-r-- 2 behera behera  660 Apr 22 16:27 six.c
7346039 -rw-rw-r-- 1 behera behera 1228 Apr 22 16:10 three.c

$ ./a.out three.c six.c
Before Copying ...
Access Time Sun Apr 22 16:10:48 2018

Modification Time Sun Apr 22 16:10:48 2018

After Copying ...
Access Time Sun Apr 22 16:27:25 2018

Modification Time Sun Apr 22 16:27:25 2018

$ ls -li three.c six.c
7346518 -rw-rw-r-- 2 behera behera  660 Apr 22 16:27 six.c
7346039 -rw-rw-r-- 1 behera behera 1228 Apr 22 16:27 three.c
```

8. Write a C program to illustrate effect of setjmp and longjmp functions on register and volatile variables.

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

static void f1(int, int, int, int);

static jmp_buf jmpbuffer;
static int globval;

int main(void)
{
            int autoval;
            register int regival;
            volatile int volaval;
            static int statval;
globval = 1; autoval = 2; regival = 3; volaval = 4; statval = 5;

if (setjmp(jmpbuffer) != 0)
{
   printf("after longjmp:\n");
   printf("globval = %d, autoval = %d, regival = %d, volaval = %d, statval =
%d\n", globval, autoval, regival, volaval, statval);
   exit(0);
}
// Change variables after setjmp, but before longjmp.

globval = 95; autoval = 96; regival = 97; volaval = 98; statval = 99;

f1(autoval, regival, volaval, statval);     /* never returns */
exit(0);
}

static void f1(int i, int j, int k, int l)
{
   printf("in f1():\n");
   printf("globval = %d, autoval = %d, regival = %d, volaval = %d, statval =
%d\n", globval, i, j, k, l);
   globval=10000;
   longjmp(jmpbuffer, 1);
}

//checks for setjmp() returns 0  ( the return is from a direct invocation)
// it returns a non-zero value when it is a call from longjmp, setjmp
// goes to f1(), which moves the execution to setjump
```

// Removed the unnecessary f2 function
https://docs.google.com/document/d/18BwpuvW-4HtDThdJNq1Xtq-jyBXvxLW08NH_pXcp_cg/edit refer this for orignial program

```
$ ./a.out
in f1():
globval = 95, autoval = 96, regival = 97, volaval = 98, statval = 99
after longjmp:
globval = 10000, autoval = 96, regival = 97, volaval = 98, statval = 99
```

9. C program to simulate copy command by accepting the filenames from command line. Report all errors.

```
                    #include<stdio.h>
#include<fcntl.h>
#include<unistd.h>
#include<stdlib.h>
int main(int argc, char *argv[])
{
          char buf[100];
          int fd1,fd2;
          off_t size,ret,set;
          ssize_t readdata,writedata;

          if(argc<3)
          printf("TOO FEW ARGUMENTS");

          if((fd1=open(argv[1],O_RDONLY)) == -1) //Open file 1
          printf("ERROR IN OPENING FILE: FILE DOES NOT EXIST \n");
          else
          printf("FILE 1 OPENED SUCCESSFULLY \n");

//open file 2 in RW mode, truncate its length to 0, create the file if it does
not exist, 0666 is the access permission for the created file. order is
important.
          if((fd2=open(argv[2],O_WRONLY | O_CREAT | O_TRUNC, 0666))
== -1)
           printf("ERROR IN OPENING FILE");

          else
           printf("FILE 2 OPENED SUCCESSFULLY \n");

size=lseek(fd1,0L,SEEK_END);
//obtain the size of file 1 using lseek

if(size==-1)
   printf("ERROR: COULD NOT OBTAIN FILE SIZE \n");

else
   printf("FILE SIZE OF FILE 1 OBTAINED \n");

ret=lseek(fd1,0L,SEEK_SET);
//change the current pointer to the beginning of the file

if(ret==-1)
   printf("RETRACE FAILED \n");

if((readdata=read(fd1,buf,size)) == -1)
   printf("ERROR IN READING FILE CONTENTS \n");

if((writedata=write(fd2,buf,size)) != size)
   printf("ERROR IN COPYING FILE");

else
   printf("FILE COPIED SUCCESSFULLY");

return 0;
}
```

./a.out:

```
//Create two files - copy contents of file 1 into file 2
$ vi magic.txt
$ vi tricks.txt
$ ./a.out magic.txt tricks.txt
```

```
FILE 1 OPENED SUCCESSFULLY
FILE 2 OPENED SUCCESSFULLY
FILE SIZE OF FILE 1 OBTAINED
FILE COPIED SUCCESSFULLY
```

10. Write a C program to avoid zombie status of a process.

```
#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>
#include<sys/wait.h>
#include<stdlib.h>
int main(void)
{
pid_t pid;

if ((pid = fork()) < 0)
            printf("fork error");

else if (pid == 0)
            {
            /* first child */
            if ((pid = fork()) < 0)
                        printf("fork error");
            else if (pid > 0)
                        exit(0);

            sleep(2);
            printf("second child, parent pid = %ld\n", (long)getppid());
            exit(0);
            }

if (waitpid(pid, NULL, 0) != pid)
printf("waitpid error");

exit(0);
}
```

./a.out:

```
//notice how it moves to the next line
:~$ ./a.out
:~$ second child, parent pid = 1
```

// should be adopted by init
// print all pid's and check.
//init has a different pid for different systems

11. Write a C program to demonstrate race condition among parent and child processes.

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

```
#include<sys/types.h>
#include<unistd.h>
#include<stdlib.h>

static void charatatime(char *);
int main(void)
{
pid_t pid;

if ((pid = fork()) < 0)
printf("fork error");

else if (pid == 0)
charatatime(" **child child child child child child child child child ** \n");

else
charatatime(" PARENT PARENT\n");
exit(0);
}

static void charatatime(char *str)
{
char *ptr; int c;
setbuf(stdout, NULL); /* set unbuffered */
for (ptr = str; (c = *ptr++) != 0; )
        putc(c, stdout);
}
```

output:

```
 PARENT P A*R*EcNhTi
ld child child child child child child child child **
//or could even be
PARENT PARENT
 **child child child child child child child child child **
```

12. Write a C program such that it initializes itself as a daemon Process.

13. Write a C program using sigaction system call which calls a signal handler on SIGINT signal and then reset the default action of the SIGINT

signal

```
#include <stdio.h>
#include <unistd.h>
#include <signal.h>

struct sigaction sig;

void handler(int val)
{
        printf("Interrupt Received!\n");
        sig.sa_handler = SIG_DFL;
        sigaction(SIGINT,&sig,0);
}

int main()
{
        sig.sa_flags = 0;
        sigemptyset(&sig.sa_mask);
        sigaddset(&sig.sa_mask,SIGINT); // listen only for SIGNIT
        sig.sa_handler = handler;

        sigaction(SIGINT,&sig,0);

        while(1)
        {
        printf("Do not press Ctrl+C \n");
        sleep(1);
        }
}
//press ctrl+c for the interrupt
```

./a.out

```
Do not press Ctrl+C
Do not press Ctrl+C
Do not press Ctrl+C
Do not press Ctrl+C
^CInterrupt Received!
Do not press Ctrl+C
Do not press Ctrl+C
Do not press Ctrl+C
^C
//Ctrl+c for interrupt
//stops after two interrupts
```

14. Write a C program (use signal system call)
i. which calls a signal handler on SIGINT signal and then reset the default action of the SIGINT signal
ii. Which ignores SIGINT signal and then reset the default action of SIGINT signal

```
#include <stdio.h>
#include <unistd.h>
#include <signal.h>

void callback()
{
        printf("Interrupt Received !\n");
        (void)signal(SIGINT,SIG_DFL);
}
int main()
{
        int ch,i=0;
        printf("Enter choice\n");
        scanf("%d",&ch);

        switch(ch)
        {
        case 1 :  (void)signal(SIGINT,callback); //shows the interrupt
                  break;
        case 2 :  (void)signal(SIGINT,SIG_IGN); //ignores the interrupt
                  break;
        }
        while(1)
        {
        sleep(1);
        printf("Press CTRL+C ...\n");
        i++;
        if(i == 10 && ch == 2)
        (void) signal(SIGINT,SIG_DFL);
        }
        return 0;
}
```

```
./a.out
Enter choice
1
Press CTRL+C ...
Press CTRL+C ...
^CInterrupt Received !
Press CTRL+C ...
Press CTRL+C ...
Press CTRL+C ...
Press CTRL+C ...
^C
****And*****
Enter choice
2
Press CTRL+C ...
Press CTRL+C ...
^C
Press CTRL+C ...
^C
Press CTRL+C ...
Press CTRL+C ...
^Z
[2]+  Stopped
```