Program 2.1 Write a C program to read from file and write to another file.

AIM: To write a C program to copy content from one file to another file.

Program:

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

#define BUFF_SIZE 1000

int main(void)
{

  int n,fd1,fd2;
  char buff[BUFF_SIZE];

   //open the file for reading
  fd1 = open("testfile.txt",O_RDWR,0644);

  //read the data from file
  n=read(fd1,buff,BUFF_SIZE);


// creating a new file using open.
      fd2=open("fileforcopy.txt", O_CREAT | O_RDWR, 0777);

       //writting data to file (fd)
      if( write(fd2, buff, n) == n)
      printf("file copying is successful. and the data is:\n");

     //Write to display (1 is standard fd for output device)
       write(1, buff, n);

       //closing the files
       int close(int fd1);
       int close(int fd2);

       return 0;

}
```

INPUT &OUTPUT:

```
testfile.txt file copied to fileforcopy.txt
file copying is successful. and the data is: hello mgit
```

Program 2.2. Program using lseek() system call that reads 10 characters from file "seeking" and print on screen. Skip next 5 characters and again read 10 characters and write on screen.

AIM: To write a C program to demonstrate the usage of lseek() system call.
Description:

lseek (C System Call): lseek is a system call that is used to change the location of the read/write pointer of a file descriptor. The location can be set either in absolute or relative terms.

off_t lseek(int *fildes*, off_t *offset*, int *whence*);

The *lseek*() function shall set the file offset for the open file description associated with the file descriptor *fildes*, as follows:

- If *whence* is SEEK_SET, the file offset shall be set to *offset* bytes.
- If *whence* is SEEK_CUR, the file offset shall be set to its current location plus *offset*.
- If *whence* is SEEK_END, the file offset shall be set to the size of the file plus *offset*.

The symbolic constants SEEK_SET, SEEK_CUR, and SEEK_END are defined in <*unistd.h*>. Upon successful completion, the resulting offset, as measured in bytes from the beginning of the file, shall be returned. Otherwise, (**off_t**)-1 shall be returned, *errno* shall be set to indicate the error, and the file offset shall remain unchanged.

Program:

```
#include<unistd.h>
#include<fcntl.h>
#include<sys/types.h
#include<sys/stat.h>
#int main()
{
        int n,f;
        char buff[10];
        f=open("seeking",O_RDWR);
        read(f,buff,10);
        write(1,buff,10);
        read(f,buff,10);
        write(1,buff,10);
}
```

Pre-requisite: Create a file "seeking" and write "1234567890abcdefghijxxxxxxxxxx" into it.

Output:

The output will be the first 10 characters "1234567890" followed by "fghijxxxxx". The inbetween 5 characters are skipped because we used lseek to reposition the pointer 5 characters ahead from the current (SEEK_CUR) position.

Program 2.3 : C Program for opendir(), readdir() and closedir system calls

AIM: To write a C program to demonstrate the usage of opendir(), readdir() and closedir() system call.

Description:

The *opendir*() function shall open a directory stream corresponding to the directory named by the *dirname* argument. Directory entries represent files or subdirectory.

```
#include <dirent.h>
DIR *opendir(const char *dirname);
```

- Upon successful completion, *opendir*() shall return a pointer to an object of type **DIR**. Otherwise, a null pointer shall be returned and *errno* set to indicate the error.

The *readdir*() function shall return a pointer to a structure representing the directory entry at the current position in the directory stream specified by the argument *dirp*, and position the directory stream at the next entry. It shall return a null pointer upon reaching the end of the directory stream.

* readdir(DIR *dirp);

* Upon successful completion, *readdir*() shall return a pointer to an object of type **struct dirent**. When an error is encountered, a null pointer shall be returned and *errno* shall be set to indicate the error.

The *closedir*() function shall close the directory stream referred to by the argument *dirp*. Upon successful completion, *closedir*() shall return 0; otherwise, -1 shall be returned and *errno* set to indicate the error.

PROGRAM:

```
#include<stdio.h>
#include<dirent.h>
struct dirent *dptr;
int main(int argc, char *argv[])
{
  char buff[100];
  DIR *dirp;
  printf("\n\n ENTER DIRECTORY NAME");
  scanf("%s", buff);
  if( ( dirp=opendir( buff ) ) == NULL)
  {
    printf("The given directory does not exist");
    exit(1);
  }
  while(dptr = readdir ( dirp ) )
  {
    printf("%s\n",dptr->d_name);
  }  closedir(dirp); }
```

Program 2.3: Write a C program to print file status information using stat function.

AIM: To write a C program to demonstrate the usage of stat() system call.

Description:

Stat system call is a system call in Linux to check the status of a file such as to check when the file was accessed. The stat() system call actually returns file attributes. The file attributes of an inode are basically returned by Stat() function. An inode contains the metadata of the file. An inode contains: the type of the file, the size of the file, when the file was accessed (modified, deleted) that is time stamps, and the path of the file, the user ID and the group ID, links of the file, and physical address of file content.

To use the stat system call in C programming language we should include the following header file:

```
#include <sys/stat.h>
int stat(const char *path, struct stat *buf)
```

- The return type of the function in int, if the function is executed successfully, 0 is returned if there are any errors, -1 will be returned.
- **const char *path** specifies the name of the file.
- stat structure The data or information about the file is stored which uses a pointer named **buf,** which is passed in as a paramteter and filled in during the execution of the call and readable by the user after the call.

Program :

```
#include<stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <dirent.h>
#include<unistd.h>
struct stat statbuf;
char dirpath[256];

int main(int argc, char *argv[])
{
   // getcwd is to get the name of the current working directory if found
getcwd(dirpath,256);
DIR *dir = opendir(dirpath);
struct dirent *dp;

for (dp=readdir(dir); dp != NULL ;  dp=readdir(dir))

 {
 stat(dp->d_name, &statbuf);
 printf("the file name is %s \n", dp->d_name);
 printf("dir = %d\n", S_ISDIR(statbuf.st_mode));
 printf("file size is %ld in bytes \n", statbuf.st_size);
```

```
    printf("last modified time is %ld in seconds \n", statbuf.st_mtime);
    printf("last access time is %ld in seconds \n", statbuf.st_atime);
    printf("The device containing the file is %ld\n", statbuf.st_dev);
    printf("File serial number is %ld\n\n", statbuf.st_ino);
    }
}
```

OUTPUT:

```
the file name is fileone.txt
dir = 0
file size is 31 in bytes
last modified time is 1581656490 in seconds
last access time is 1581656969 in seconds
The device containing the file is 2053
File serial number is 2099241

the file name is filestatus.c
dir = 0
file size is 772 in bytes
last modified time is 1581146159 in seconds
last access time is 1581920990 in seconds
The device containing the file is 2053
File serial number is 2099181
```