1a)
```
#include <stdlib.h>
#include <stdio.h>
#include <fcntl.h>
#include <string.h>
#include <sys/stat.h>
#include <unistd.h>

int main(int argc, char *argv[])
{
        int source, dest, n;
        char buf;
        int filesize;
        int i;

        if (argc != 3)
        {
        fprintf(stderr, "usage %s <source> <dest>", argv[0]);
        exit(-1);
        }

        if ((source = open(argv[1], O_RDONLY)) < 0)
        {
        fprintf(stderr, "can't open source\n");
        exit(-1);
        }

        if ((dest = open(argv[2], O_WRONLY | O_CREAT | O_TRUNC)) < 0)
        {
        fprintf(stderr, "can't create dest\n");
        exit(-1);
        }

        filesize = lseek(source, (off_t)0, SEEK_END);
        printf("Source file size is %d\n", filesize);

        for (i = filesize - 1; i >= 0; i--)
        {
        lseek(source, (off_t)i, SEEK_SET);

        if ((n = read(source, &buf, 1)) != 1)
        {
        fprintf(stderr, "can't read 1 byte");
        exit(-1);
        }
```

```
	if ((n = write(dest, &buf, 1)) != 1)
	{
	fprintf(stderr, "can't write 1 byte");
	exit(-1);
	}
	}
	write(STDOUT_FILENO, "DONE\n", 5);
	close(source);
	close(dest);

	return 0;
}
```

1b)

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/wait.h>

int main()
{
	int fd = open("test.txt", O_RDWR);
	if (fd == -1)
	{
	perror("open");
	return 1;
	}
	pid_t pid = fork();
	if (pid == -1)
	{
	perror("fork");
	return 1;
	}
	else if (pid == 0)
	{
	char buffer[10];
	read(fd, buffer, 5);
	buffer[5] = '\0';
	printf("Child read: %s\n", buffer);
	}
	else
	{
	wait(NULL);
```

```
        char buffer[10];
        read(fd, buffer, 5);
        buffer[5] = '\0';
        printf("Parent read: %s\n", buffer);
        }
        close(fd);
        return 0;
}
```

```
2a)#include <unistd.h>
#include <stdio.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <time.h>

int main(int argc, char *argv[])
{
        if (argc != 2){
        printf("Usage: %s <filename>\n", argv[0]);
        return 1;
        }

        struct stat fileStat;

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

        printf("File Name: %s\n", argv[1]);
        printf("File Size: %lu bytes\n", fileStat.st_size);
        printf("Number of hard Links: %lu\n", fileStat.st_nlink);
        printf("File inode: %lu\n", fileStat.st_ino);
        printf("File Permissions: %o\n", fileStat.st_mode);
        printf("Last Access Time: %s", ctime(&fileStat.st_atime));
        printf("Last Modification Time: %s", ctime(&fileStat.st_mtime));

        return 0;
}
```

```
2b)#include <stdio.h>
#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
```

```
int main(int argc, char *argv[])
{
        if (argc == 3)
        {
        printf("Hard linking %s and %s\n", argv[1], argv[2]);
        if (link(argv[1], argv[2]) == 0)
        printf("Hard link created successfully.\n");
        else
        printf("Error creating hard link");
        }
        else if (argc == 4)
        {
        printf("Soft linking %s and %s\n", argv[1], argv[2]);
        if (symlink(argv[1], argv[2]) == 0)
        printf("Soft link created successfully.\n");
        else
        printf("Error creating soft link");
        }

        return 0;
}
```

3a)#include <stdio.h>

```
#include <fcntl.h>

#include <unistd.h>

#include <dirent.h>

int main()
{
        DIR *dp;
        struct dirent *dir;
        int fd, n;
        dp = opendir("."); // open current directory
        if (dp)
        {
        while ((dir = readdir(dp)) != NULL)
        {
        fd = open(dir->d_name, O_RDWR, 0777);
        n = lseek(fd, 0, SEEK_END);
        if (!n)
```

```
            {
            unlink(dir->d_name);
            }
            }
            }
            return 0;
}

3b)#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)
            {
            printf("--inode--mode-- uid--guid--access_time--Filename\n");
            while (dir = readdir(dp))
            {
            stat(dir->d_name, &mystat);

            printf("\n%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);
            }
            }
            return 0;
}

4a)#include <stdio.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
#include <utime.h>
#include <time.h>

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

```
            struct stat statbuf_1;
            struct stat statbuf_2;
            struct utimbuf times;

            if (stat(argv[1], &statbuf_1) < 0) /*Destination file status*/
            printf("Error!\n");

            if (stat(argv[2], &statbuf_2) < 0) /* Source file status*/
            printf("Error!\n");

            printf("Before Copying ...\n");
            printf("Access Time %s\nModification Time%s\n", ctime(&statbuf_1.st_atime),
ctime(&statbuf_1.st_mtime));

            times.modtime = statbuf_2.st_mtime;
            times.actime = statbuf_2.st_atime;
            if (utime(argv[1], &times) < 0)
            printf("Error copying time \n");

            if (stat(argv[1], &statbuf_1) < 0)
            printf("Error!\n");

            printf("After Copying ...\n");
            printf("Access Time %s\nModification Time%s\n", ctime(&statbuf_1.st_atime),
ctime(&statbuf_1.st_mtime));
            return 0;
}
```

```
4b)#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int main()
{
            int fd1 = 0, fd2 = 0;
            char buf[50];
            if ((fd1 = open("test.txt", O_RDWR, 0)) < 0)
            printf("file open error");

            fd2 = dup(fd1);
            printf("%d %d \n", fd1, fd2);
            read(fd1, buf, 10);
            lseek(fd2, 0L, SEEK_END);
```

```
        write(fd2, buf, 10);
        printf("%s\n", buf);
        return 0;
}

5a)#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
        int i;
        struct stat buf;
        char *ptr;
        for (i = 1; i < argc; i++)
        {
        printf("%s: ", argv[i]);
        if (lstat(argv[i], &buf) < 0)
        {
        printf("lstat error");
        continue;
        }
        if (S_ISREG(buf.st_mode))
        ptr = "regular";
        else if (S_ISDIR(buf.st_mode))
        ptr = "directory";
        else if (S_ISCHR(buf.st_mode))
        ptr = "character special";
        else if (S_ISBLK(buf.st_mode))
        ptr = "block special";
        else if (S_ISFIFO(buf.st_mode))
        ptr = "fifo";
        else if (S_ISLNK(buf.st_mode))
        ptr = "symbolic link";
        else if (S_ISSOCK(buf.st_mode))
        ptr = "socket";
        else
        ptr = "** unknown mode **";
        printf("%s\n", ptr);
        }
        exit(0);
}
```

```
5b)#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <string.h>

int main() {
        // dup() demonstration
        int fd1 = open("dup_test.txt", O_WRONLY | O_CREAT | O_TRUNC, 0644);
        int fd2 = dup(fd1);
        printf("dup(): Original fd: %d, New fd: %d\n", fd1, fd2);
        write(fd1, "Line from fd1\n", 14);
        write(fd2, "Line from fd2\n", 14);
        close(fd1);
        close(fd2);

        // dup2() demonstration
        int fd3 = open("dup2_test.txt", O_WRONLY | O_CREAT | O_TRUNC, 0644);
        int fd4 = 10;  // Specific fd number we want to use
        dup2(fd3, fd4);
        printf("dup2(): Original fd: %d, Specified fd: %d\n", fd3, fd4);
        write(fd3, "Line from fd3\n", 14);
        write(fd4, "Line from fd4\n", 14);
        close(fd3);
        close(fd4);

        return 0;
}
```

6a-chmod)

```
#include <stdio.h>
#include <fcntl.h>
#include <stdlib.h>
#include <sys/stat.h>
int main()
{
        struct stat statbuf;
        if (stat("foo", &statbuf) < 0)
        printf("stat error for foo");
        if (chmod("foo", (statbuf.st_mode & ~S_IXGRP) | S_ISGID) < 0)
        printf("chmod error for foo");
        /* set absolute mode to "rw-r--r--" */
        if (chmod("bar", S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH) < 0)
        printf("chmod error for bar");
        exit(0);
}
6a-unmask)
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/stat.h>
#include <fcntl.h>
#define RWRWRW (S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH | S_IWOTH)

int main()
{
        umask(0);
        if (creat("foo", RWRWRW) < 0)
        printf("creat error for foo");
        umask(S_IRGRP | S_IWGRP | S_IROTH | S_IWOTH);
        if (creat("bar", RWRWRW) < 0)
        printf("creat error for bar");
        exit(0);
}

6b)#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
int main()
{
        int file = 0, n;
        char buffer[25];
        if ((file = open("test.txt", O_RDONLY)) < -1)
        printf("file open error \n");
        if (read(file, buffer, 20) != 20)
        printf("file read operation failed\n");
        else
        write(STDOUT_FILENO, buffer, 20);
        printf("\n");
        if (lseek(file, 10, SEEK_SET) < 0)
        printf("lseek operation to beginning of file failed\n");
        if (read(file, buffer, 20) != 20)
        printf("file read operation failed\n");
        else
        write(STDOUT_FILENO, buffer, 20);
        printf("\n");

        if (lseek(file, 10, SEEK_CUR) < 0)
        printf("lseek operation to beginning of file failed\n");
        if (read(file, buffer, 20) != 20)
        printf("file read operation failed\n");
        else
        write(STDOUT_FILENO, buffer, 20);
        printf("\n");

        if ((n = lseek(file, 0, SEEK_END)) < 0)
        printf("lseek operation to end of file failed\n");
        printf("size of file is %d bytes\n", n);
        close(file);
        return 0;
}
```

```
7a)#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/types.h>
int main()
{
        int st;
        pid_t pid1 = fork();
        pid_t pid2 = fork();
        if (pid1 == 0)
        {
        printf("first pid:%d\n", getpid());
        sleep(2);
        exit(0);
        }
        if (pid2 == 0)
        {
        printf("second pid:%d\n", getpid());
        sleep(4);
        exit(0);
        }
        wait(&st);
        printf("first wait\n");
        sleep(1);
        waitpid(pid2, &st, 0);
        printf("second wait\n");
        return 0;
}
```

7b)

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/types.h>
int main()
{
        int st;
        pid_t pid1 = fork();
        pid_t pid2 = fork();
        if (pid1 == 0)
        {
        printf("first pid:%d\n", getpid());
        sleep(2);
        exit(0);
        }
        if (pid2 == 0)
        {
        printf("second pid:%d\n", getpid());
        sleep(4);
        exit(0);
        }
        wait(&st);
        printf("first wait\n");
        sleep(1);
        waitpid(pid2, &st, 0);
        printf("second wait\n");
        return 0;
}
```